DOS/V プログラミング技法

98からDOS/Vの世界へ

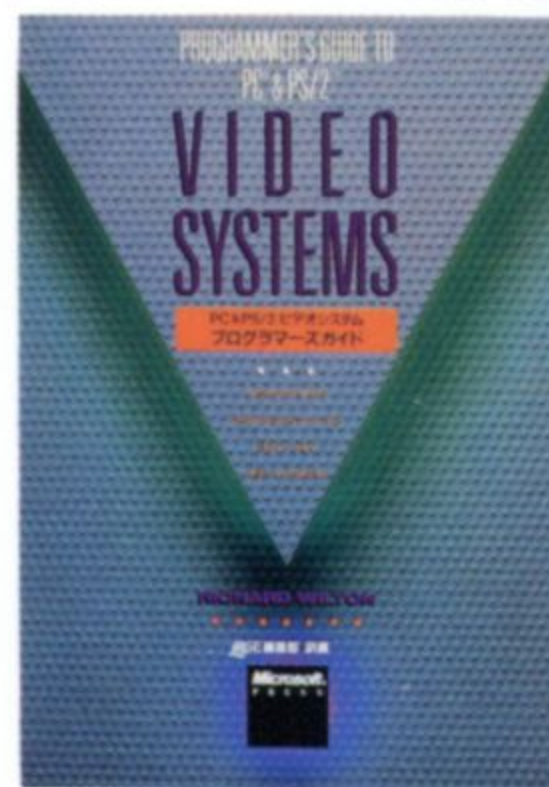
杉浦明美・柴﨑忠生 共著

programmer's page

3.5"
2DD FD付

DOS/V BIOSプログラミングのすべてを、
MASM6.0のコーディング例を中心に詳説。
98との相違についても解説。

CLOSE COVER BEFORE STRIKING

SE
SHOEISHA

DOS/V プログラミング技法

programmer's page

98からDOS/Vの世界へ

杉浦明美・柴﨑忠生 共著

CLOSE COVER BEFORE STRIKING

SE
SHOEISHA

まえがき

昔からIBM PCを所有する夢があったのですが，日本IBMから販売されている機種は非常に高価であり，また，海外から直輸入しても，日本語は全く使えず，ただの遊びになってしまうとの気持ちがあり，購入には結びつきませんでした．一昨年（1991年）にDOS/Vが発表された当初は，仕事で使用していたPS/55と比較すると，フォントも汚く，まだまだ使用にたえないと思っていました．しかし，一昨年後半から秋葉原でAT互換機が販売されるようになり，IBM PCが現実のものとして身近になってきて，1991年10月に秋葉原で見せられた高解像度・高速のDOS/Vパソコンを不安のなか，購入してしまいました．最初のころはソフトウェアもあまりなく，本当に使いこなせるのか，不安な点もあったのですが，いくつかのソフトウェアを作っていくうちに，しだいにのめり込んでしまったのです．

とくに，私の主人がNIFTY Serveに加入していたこともあり，PC-9801のフリーソフトウェアをDOS/Vに数多く移植してくれました．また，昨年からはフリーソフトウェアとしてハイテキスト表示が可能となってからは，ソースプログラムの見通しの良さからプログラム開発はすべてDOS/V環境に移行してしまい，PC-9801はしだいに使わなくなってしまいました．

昨年末に，友人である柴崎氏とOS/2やDOS/Vに関する本を出さないかという話しをしているうちに，この本を作成する話しがとんとんと進んでしまいました．本当なら，年初にも出したいとの予定でしたが，なかなか執筆が進まず，出版社の方にも御迷惑をかけてしまいました．

DOS/V環境でのプログラミングを行う際に，なかなか手頃な資料が見つからず，IBM PC系の資料をいろいろと漁りました．しかし，どの資料もDOS/Vに関しては一切書かれておらず，本当にその機能が使用できるのかを確認する必要性がありました．

本書は，DOS/V用プログラムを作成するための知識・技法に関して書かれています．私や主人が，いままで，いろいろなフリーソフトウェアや業務プログラムを作成して習得したノウハウを公開したつもりです．本書を参考に，よりよいプログラムを作成してもらいたいと思っています．

最後になりましたが，本書の出版にあたり，スケジュールを伸ばしていただいた翔泳社の上田さん，いろいろ技術的アドバイスをしてくれた主人，ありがとうございました．

1993年5月
杉浦　明美

目　次

第0章　はじめに　1

0.1　DOS/Vとは何なのか ……2

0.2　使用したハードウェアと開発環境の概要 ……3

0.3　アセンブラマクロ ……3

0.4　PC-9801との違い ……5

画面制御
キーボード
RS-232C
マウス
タイマ
その他の周辺機器
漢字コード

0.5　参考文献 ……9

第1章　DOS/Vでのプログラミングとは　11

1.1　プログラミングのレベルを決める ……12

DOSの標準出力だけを使用
INT 29hを使用
ビデオBIOSを使用
直接，VGAなどのハードウェアを操作

1.2　動作環境の判定方法 ……14

1.3　英語モードと日本語モードの切り替え ……17

DBCSベクタテーブル
コードページ

CONTENTS

CONTENTS

第2章　画面制御編　21

2.1　基礎知識 …… 22

$FONT.SYSと$DISP.SYSのはたらき

V-Text（Variable-Text）

2.2　ビデオモード …… 26

ビデオモード番号

ビデオBIOSを使用した画面モードの取得と設定

2.3　エスケープシーケンス文字を使用した画面表示 …… 34

2.4　カーソル制御 …… 37

2.5　文字の読み取り …… 39

2.6　文字の書き込み …… 41

2.7　仮想VRAMへの直接表示 …… 43

2.8　スクロール …… 46

2.9　パレットの取得と設定 …… 51

2.10　フォントの取得と設定 …… 55

2.11　V-Textインターフェース …… 59

Super Drivers

DOS/V Extension Ver.1.0

非公開機能

2.12　V-Text対応のプログラムを作る …… 65

2.13　グラフィック処理 …… 71

第3章　キーボード編　77

3.1　基礎知識 …… 78

キーボードの種類

キーボード割り込み処理

特殊キーの処理

CONTENTS

3.2　シフト状態を制御する（AH=02h,12h） ……87

シフト状況の取得

シフト状態の制御

漢字シフトの制御

3.3　キーボードからの入力（AH=00h,01h,10h,11h） ……95

次文字の読み取りと文字の入力状況

キーリピート速度の変更（AX=0305h）

3.4　キー入力バッファへの書き込み（AH=05h） ……99

3.5　走査コードの変換（AH=4Fh,INT 15h） ……100

第4章　RS-232C編　103

4.1　基礎知識 ……104

ハードウェア割り込み

関連ハードウェア

割り込みベクタ

4.2　初期化と終了処理 ……109

割り込みベクタおよびI/Oポートのベースアドレスの決定

割り込みの禁止

ライン制御レジスタの設定

割り込みベクタの保存と書き換え

FIFOモードの設定と割り込みの許可

割り込みコントローラの割り込みマスクの指定

修了処理

4.3　信号線の制御 ……115

モデム制御レジスタ

モデムステータスレジスタ

4.4　送受信処理 ……118

受信データレディ割り込み

送信レジスタ空き割り込み

割り込み処理からの復帰

4.5　ブレーク信号の送出 ……122

第5章　マウス編　125

5.1　基礎知識 …… 126

マウスインターフェースを使用可能にする

マウスインターフェースの使用上の考慮点

マウスインターフェースの使用上の予備知識

5.2　マウスインターフェースを使う …… 130

マウスインターフェースの呼び出し方法

サンプルプログラムについて

マウスインターフェースAPI

第6章　タイマ編　155

6.1　基礎知識 …… 156

6.2　55ミリ秒より短い間隔の割り込みが必要な場合 …… 158

タイマ割り込みルーチンの設定（StartRealTime）

タイマ割り込みハンドラでの処理（RealTimeHandler）

タイマ割り込みルーチンの解除（StopRealTimer）

6.3　システムタイマとリアルタイムクロック …… 160

6.4　サウンド …… 165

第7章　その他の周辺機器編　169

7.1　パラレルポート …… 170

7.2　ドライブ制御 …… 171

付　録　資料編　175

資料編1　BIOS割り込み一覧 …… 176

資料編2　BIOSワークエリア …… 188

資料編3　I/Oポート …… 193

CONTENTS

CONTENTS

資料編４　割り込み一覧 ……………………………………………198

添付ディスクについて ……………………………………………201

さくいん ……………………………………………203

図一覧 ……………………………………………207

表一覧 ……………………………………………209

リスト一覧 ……………………………………………211

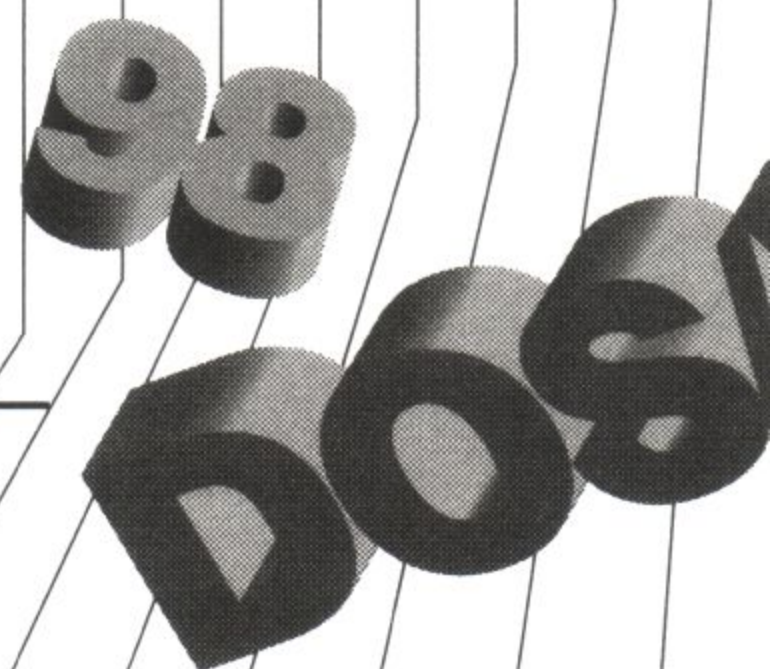
第0章

98からDOS/Vの世界へ

98 DOS/V

一昨年（1991年）にIBM DOS/Vが発売され，これまで英語環境下でしか使用できなかったIBM PC/AT互換機（以下AT互換機と省略）が，日本語パソコンとして使用できるようになりました．低価格・高速のパソコンとして，昨年後半から非常に話題となってきています．

これらAT互換機，通称いわゆるDOS/Vパソコンは，はじめのうちこそサポート体制などに関していろいろといわれることがありましたが，現在ではメーカーやショップの対応も改善されて，国産パソコンと比較しても遜色ありません．また，DOS/Vを推進している日本IBMもユーザーの意見を積極的に吸い上げ，さらにはNIFTY-Surveで，FIBMFEEL会議室の開催をサポートするなど，メーカーとしての対応にも好感がもてます．何よりも，DOS/Vパソコンを利用することによって，海外と時差がなく，おもしろいハードウェアとソフトウェアを使用できるようになったことは，いままで以上にパソコンとプログラミングに対する夢を広げてくれたといえるでしょう．

98 DOS/V

はじめに

0-1 DOS/Vとは何なのか

ところで，現在話題になっているDOS/Vとはいったい何物なのでしょうか．基本的には，海外で主流となっているAT互換機のハードウェアをほとんどそのまま使用し（基本的に異なるのはキーボードだけですが，これもUS版の101キーボードでも使用できます），ソフトウェアだけで日本語表示を可能としたOSです．では，DOS/Vは特別な日本語DOSがあるのかというと，そうではありません．DOS/Vは単純にPC-DOSに日本語フォント処理と表示ドライバを付加しただけのものです．当然のことながら，ファイル名などの日本語処理およびメッセージなどの日本語化なども行われていますが，根本的にDOSに対する大幅な変更は加えられていないため，日本語を表示することによって，DOSそのものに問題が起きる可能性は少ないといえるでしょう．

DOS/Vの日本語表示の基本的なしくみは，「第2章　画面制御編」で詳しく解説しますが，日本語フォントの扱いおよび画面表示処理は，VGAグラフィックを使用して，すべてソフトウェアで行っています．ただし，アプリケーションプログラムから見れば，仮想VRAMをもっていますので，いままでのPC-9801と似たようなコーディングが可能となっています．また，AT互換機ではビデオシステム自身が本体とは独立しており，ユーザーが必要とあらば，これも取り替えることが可能となっています．もともとのビデオBIOSにそういった拡張性が考慮されており，ビデオシステムそのものも過去のビデオシステムと上位互換性を保っていますので，必要に応じて，高速・高解像度のビデオシステムと交換することができるのです．

DOS/V環境では，VGAと上位互換性をもつビデオシステム（最近では，SVGAやXGAと呼ばれているもの）であれば，使用することができます．ただし，ひとくちにSVGAといっても，その中心となるチップによって機能・解像度などが異なりますので，注意は必要です．最近の傾向としては，S3社のチップがVGAモードおよびWindowsのアクセレータ機能の処理が速いので，よく利用されています．

XGAやSVGAは，いままでのVGA用の640×480ドットの解像度のほかに，800×600や1024×768，およびそれ以上の解像度をもっています．最近ではこの高解像度を使用した高品位，高密度の画面表示を行うビデオドライバが登場し，その需要はさらに高まりつつあります．DOS/Vの拡張として，日本IBMからV-Textと呼ばれる拡張ビデオドライバが発表され，これまでの80桁×25行の固定された画面の世界は，いまや大きく変わろうとしています．

DOS/Vに関しては，雑誌・単行本などが，昨年（1992年）末からいっきに販売されはじめ，プログラミング関連の資料も増えてきましたが，まだ十分とはいえません．そこで

本書では，DOS/Vでプログラミングを行ううえでの各種の注意事項にふれながら，実際のBIOSコーディング手法を中心に解説を行い，DOS/V自身の理解を深め，C言語から利用できるBIOS関数や実際のBIOS関数の使用例などを作成していきます．

0.2 使用したハードウェアと開発環境の概要

本書で紹介するプログラムは，Qualest 486/33，JCS Vintage 486DX2/80（改造版），PS/55 5551-T0B，PS/55note N23 sx，DynaBook J3100EZ（DOS/V環境）で動作確認を行っています．また，DOSもIBM DOSバージョンJ5.02/VおよびJ5.02（DOS/Vモード），DR DOS 6.0/V1.00を使用しています．

日本語ドライバとしては，それぞれのDOSでの標準の$FONT.SYS，$DISP.SYSならびにフリーソフトウェアの$FONTX V2.2，DISPV V1.53，DISPS3 V1.53および C.F. Computingが監修し，ソフトバンクより市販されているDOS/V Super Driversと日本IBM社より販売されているDOS/V Extension V1.00を使用しています．

また，DynaBook EZの場合は，東芝DOS 3.1上にフリーソフトウェアのTOP31.COM，DBCS.SYS，$FONTJ31.SYS，DISPJ.SYSの環境，および上記DOS/V Super Driversを使用して，DOS/V環境を作成しています．

コーディングサンプルに関しては，アセンブラ部分は，MASM 6.0形式で，C部分に関してはBorland Turbo-C（TC++，BC++のCモードを含む），およびMS-C 6.0でコンパイルできるようにしています．

また，MASM 6.0はDOS/V用のものは販売されていませんが，PC-DOS用またはPC-9801用の両者とも問題なくコマンドラインで使用できます．また，PC-DOS用は英語モードに切り替えれば，プログラマワークベンチ（PWB）も使用でき，価格も相対的に安いのでお薦めです．

0.3 アセンブラマクロ

今回のコーディングは，大半がMASM 6.0で記述されています．これは，BIOSまわりの処理を行う場合，どうしても直接CPUレジスタを操作することが必要となるためです．こういったコーディングをC言語でも行うことはできますが，特定のC言語への依存性が高く

なってしまい，汎用的に使えなくなってしまいます．また，プログラム自身も見通しが悪いものになってしまいます．また，アセンブラが一般的に敬遠される理由として，プログラムの生産性が悪いことがあげられると思います．しかし，MASM 6.0では構造化ステートメントも使用できるため，レジスタとか割り込みベクタなどがわかってくれば，C言語と同じような雰囲気でコーディングを行うことができます．実は筆者は，MASM 6.0が出る前もすでにフリーソフトウェアの構造化マクロを使用しており，構造化ができなければ，アセンブラコーディングをする気力もなくなってきています．ぜひ，この際に構造化アセンブラに慣れられることをお薦めします．

また，コーディングをわかりやすくするように，一部，アセンブラマクロを使用していますので，サンプルプログラムを読む際に注意してください．

代表的なアセンブラマクロには，以下の3種類があります．

(1)　割り込み呼び出し　　　　（形式：　INTV　cmd,prm）

実際に，命令コードINTV部分には，以下の命令を指定します．

VIDEO	(INT 10h)
DISK	(INT 14h)
INT15	(INT 15h)
KEYBORD	(INT 16h)
TIMER	(INT 1Ah)
MSDOS	(INT 21h)
INT2F	(INT 2Fh)
MOUSE	(INT 33h)
EMMDRV	(INT 67h)
VIDEO98	(INT 18h)

また，cmd,prmは通常，それぞれAH,ALに代入する値を指定します．ただし，両者を省略した場合は，単純に割り込み命令だけが生成されます．また，prmを省略した場合，cmdの値が100h未満の場合はAHに，100hを越す場合はAXに値が代入されます．

(2)　セグメントレジスタの代入　（形式：　MOVSEG　target,source)

このマクロは以下のように展開されます．

```
PUSH source
POP   target
```

(3)　レジスタのプッシュ　　　（形式：　PUSHM　<regs...>)
　　レジスタのポップ　　　　（形式：　POPM　<regs...>)

連続して，レジスタをPUSH，POPする命令です．<>中には複数のレジスタを“,”で区切って指定することができます．また，PUSH，POPは< >内の左側から行われていきます．

アセンブラマクロは添付ディスク中のSTD.INCにまとめていますので，詳細に関して知りたい場合にはSTD.INCを参照してください．

0.4 PC-9801との違い

つぎに，DOS/VでのプログラミングでPC-9801とは異なる特徴的な部分を説明しておきます．

画面制御

◎一般的に，画面表示に関してはビデオBIOSを使用します．

◎テキスト画面やグラフィック画面は，画面モードを切り替えるしくみになっていますので，テキスト画面とグラフィック画面のスーパーインポーズはできません．

◎DOS/Vテキスト画面は仮想VRAM形式ですので，直接書き込んでも即時には画面表示は行われません．

◎ファンクションキー表示などのBASIC的な機能はありません．必要ならば自分でコーディングしてください．

◎エスケープシーケンス文字は，DOS本体（INT 29hを含む）では処理されないため，ANSI.SYSまたは同等の機能をもつデバイスドライバが必要です．

キーボード

◎キーボードBIOSを使用しても漢字が取得できますので，細かなキー入力管理を行う場合にはキーボードBIOSを使用してください．

◎ファンクションキーに関しては，特定のコードが返ってくるだけであり，PC-9801のように文字登録はできません（実際には，エスケープシーケンス文字を使用して変更することはできるのですが，使用しないのが暗黙のルールとなっています）．

RS-232C

◎IBM PC系では非同期通信専用のシリアルコントローラが使用されているため，標準構成では同期通信はできません．

◎シリアルコントローラはチップ内で回線速度を設定できますので，PC-9801と異なり，8253（プログラマブルタイマコントローラ）の制御は不要です．したがって，8250B(16550)および8259Aだけを制御することにより，シリアル通信を行うことができます．

マウス

◎PC-9801の場合には，テキストモード時でもグラフィック画面をスーパーインポーズできますので，マウスカーソルはグラフィック表示されます．しかし，DOS/Vではテキストモード時はグラフィック表示と重ね合わせることができないため，文字カーソルとなります（表0.1）．

◎マウスBIOSには表0.2のような違いがあります．この表では機能名しか記載されていないため同じ名前の機能でも，パラメータが異なったり，機能詳細が異なる場合があります．

表0.1 マウスインターフェースの比較

PC-9801マウスインターフェースと，IBM PCシリアルマウスインターフェースの比較

NEC PC-9801 MOUSE.SYS	IBM PC/AT MOUSE.COM
マウスカーソル カラー200モード： 16×16ドットグラフィックカーソル カラー400モード： 16×32ドットグラフィックカーソル	マウスカーソル テキストモード：文字カーソル グラフィックモード： 16×16ドットグラフィックカーソル
割り込み INT 33H使用	割り込み INT 33H使用

表0.2 マウスBIOSの比較

PC-9801マウスBIOSと，IBM PCシリアルマウスインターフェースBIOSの比較

NEC PC-9801 MOUSE.SYS	機能NO	IBM PC-AT MOUSE.COM
マウス機能の初期化	0	マウス機能の初期化
カーソルの表示	1	カーソルの表示
カーソルの消去	2	カーソルの非表示
カーソル位置の読み取り	3	カーソル位置とボタン状態の読み取り
カーソル位置の設定	4	カーソル位置のセット
左ボタンの押下情報の取得	5	ボタンを押した回数と最終位置の読み取り
左ボタンの解放情報の取得	6	ボタンを離した回数と最終位置の読み取り
右ボタンの押下情報の取得	7	カーソル移動範囲の設定（X方向）
右ボタンの解放情報の取得	8	カーソル移動範囲の設定（Y方向）
カーソル形状の設定	9	グラフィックカーソルの形状設定
	10	テキストカーソルの設定
マウスの移動距離の取得	11	マウスの移動距離（マウスΔ）の読み取り
ユーザー割り込みルーチンの設定	12	ユーザー割り込みの設定
	13	ライトペンエミュレーション機能開始
	14	ライトペンエミュレーション機能終了
ミッキー/ドット比率の設定	15	マウスの移動比率の設定
カーソル移動範囲の設定（X方向）	16	カーソル非表示域の設定
カーソル移動範囲の設定（Y方向）	17	
カーソル表示画面の設定	18	
	19	倍速境界値の設定
	20	ユーザー割り込みルーチンの差し替え
	21	ドライバ状態保管用バッファサイズの取得
	22	ドライバ状態の保管
	23	ドライバ状態の復元
	24	代替ユーザー割り込みルーチンの設定
	25	代替ユーザー割り込みルーチンのアドレスの取得
	26	マウス感度の設定
	27	マウス感度の取得
	29	カーソル表示ページの設定
	30	カーソル表示ページの取得
	31	マウスドライバの使用禁止設定
	32	マウスドライバの使用禁止解除
	33	マウスドライバのソフトウェアリセット

タイマ

◎タイマ割り込みには，INT 1Chを使用します（1秒間に18.2回割り込みが発生します．約55ミリ秒の間隔）．この割り込みに関しては，複数のプログラムがフックする可能性がありますので，INT 1Chをフックして使用する場合には，必ず，フック前のアドレスに制御を渡すようにしてください．

◎ハードウェアタイマはすべてシステムが使用しています．また，VSYNC割り込みもありません．定期的な間隔で処理を行いたい場合には，前述のタイマ割り込み（INT 1Ch）を使用してください．

その他の周辺機器

◎PC-9801では，起動ディスクがAドライブとなりますので，ドライブ名は一般的に不定となります．その他のMS-DOSでは，FDDがA,Bドライブ，HDDがCドライブからというように，デバイスドライバで指定する部分以外は固定となります．

◎通常，印刷エスケープシーケンス文字は，IBM5577印刷装置およびESC/P印刷装置に対応していて，デバイスドライバ（$PRNUSER.SYS，$PRNESCP.SYS）を入れ替えることにより対応します．また，ライオスから，PC-PR201対応の印刷ドライバが販売されています．

漢字コード

◎漢字コードに関しては，JIS漢字コードの改定および各メーカーごとの特殊文字などの追加があり，一部漢字コードが異なります．また，MS-DOSの場合には，実際はJISコードそのままではなく，漢字コードが半角とぶつからないようにずらして，シフトJISコードにしています．

◎個々のコード体系に関して，PC-9801の場合は，基本的にはJIS78年版（JIS C6226-1978）対応です．ただし，丸数字・罫線文字などの特殊文字や2バイト半角文字があります．EPSONのPC-9801互換機の場合には，JIS83年版（JIS X0208-1983）対応で，PC-9801が採用している丸数字・罫線文字などの特殊文字や2バイト半角も追加されています．DOS/VではIBMが採用しているIBM日本語文字セットをそのまま採用しています．このコード体系は，基本的にはJIS83年版に対応していますが，78年版から83年版でコードが入れ替わった部分に関しては，過去とのアプリケーションプログラムの移行性を重視して，入れ替えていません．

◎JIS78年版から，JIS83年版への変更点は以下の通りです．

(1)　22組の異体字の入れ替え（表0.3）

JIS改訂内容：第1・第2水準間で22組のコードポイントを入れ替え．

表0.3 入れ替えられた22組の異体字

第1水準	字形	第1水準	字形	第2水準	字形	第2水準	字形
16-19	鯵	33-08	賎	82-45	鰺	76-45	賤
18-09	鴬	36-59	壷	82-84	鶯	52-68	壺
19-34	蛎	37-55	砺	73-58	蠣	66-74	礪
19-41	撹	37-78	梼	57-88	攪	59-77	檮
19-86	竃	37-83	涛	67-62	竈	62-25	濤
20-35	潅	38-86	迩	62-85	灌	77-78	邇
20-50	諌	39-72	蝿	75-61	諫	74-04	蠅
23-59	頚	41-16	桧	80-84	頸	59-56	檜
25-60	砿	43-89	侭	66-72	礦	48-54	儘
28-41	蕊	44-89	薮	73-02	蘂	73-14	藪
31-57	靭	47-22	篭	80-55	靱	68-38	籠

表0.4 追加されたた新字体漢字4文字．旧字は84区へ移動

字形		JIS（新字）	JIS（旧字）
尭	→	22-38	堯（84-01）
槙	→	43-74	槇（84-02）
遥	→	45-58	遙（84-03）
瑶	→	64-86	瑤（84-04）

IBM改訂内容：コードポイントを入れ替えない．

(2)　追加された新字体漢字4文字／旧字は84区へ移動（表0.4）

JIS改訂内容：78年版のうち4文字を新字体に入れ替え，旧字をJIS区点の84-01から84-04に追加．

IBM改訂内容：IBMは字形を変えないという原則に従い，新字体をJIS区点の84-01から84-04に追加．

(3)　特殊文字・罫線文字71文字の追加

JIS改訂内容：特殊文字・罫線文字71文字を追加．

IBM改訂内容：すでにIBMで採用されている2文字（表0.5）を除いた，特殊文字・罫

表0.5 IBM選定文字に含まれていた2文字

JIS	IBM	字形	JIS	IBM	字形
02-44	115-21	¬	02-72	115-28	∵

表0.6 追加された特殊文字・罫線文字など69文字

JIS	字形	JIS	字形	JIS	字形
02-26	∈	02-69	√	08-10	┴
02-27	∋	02-70	∽	08-11	┼
02-28	⊆	02-71	∝	08-12	━
02-29	⊇	02-73	∫	08-13	┃
02-30	⊂	02-74	∬	08-14	┏
02-31	⊃	02-82	Å	08-15	┓
02-32	∪	02-83	‰	08-16	┛
02-33	∩	02-84	♯	08-17	┗
02-42	∧	02-85	♭	08-18	┣
02-43	∨	02-86	♪	08-19	┳
02-45	⇒	02-87	†	08-20	┫
02-46	⇔	02-88	‡	08-21	┻
02-47	∀	02-89	¶	08-22	╋
02-48	∃	02-94	◯	08-23	┠
02-60	∠	08-01	─	08-24	┯
02-61	⊥	08-02	│	08-25	┨
02-62	⌒	08-03	┌	08-26	┷
02-63	∂	08-04	┐	08-27	┿
02-64	∇	08-05	┘	08-28	┝
02-65	≡	08-06	└	08-29	┰
02-66	≒	08-07	├	08-30	┥
02-67	≪	08-08	┬	08-31	┸
02-68	≫	08-09	┤	08-32	╂

線文字69文字を追加（表0.6）.

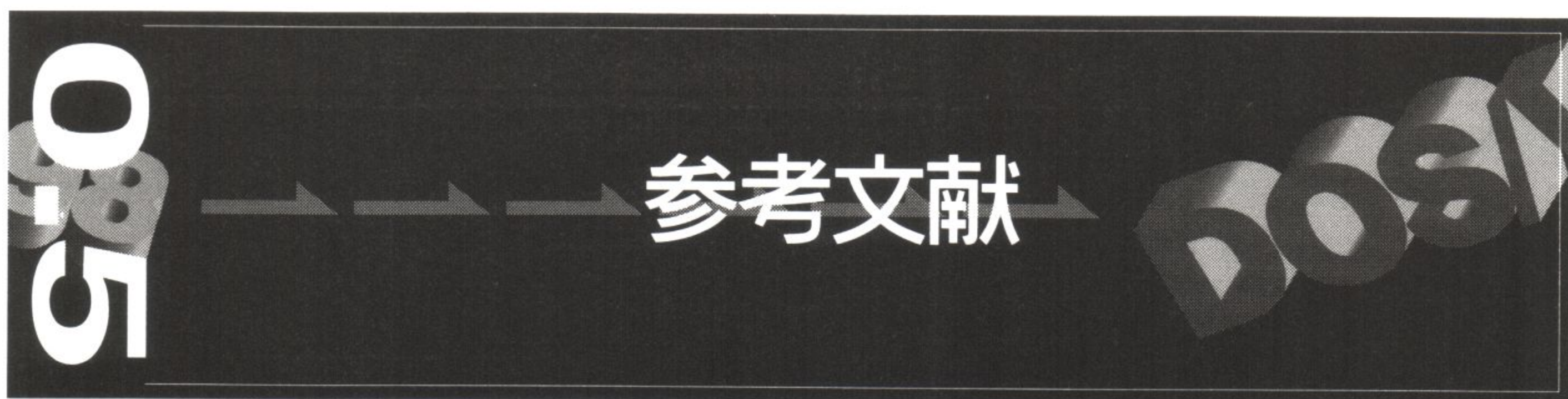
(4)　244文字の字形変更

JIS改定内容：244字の字形変更.

IBM改定内容：常用漢字表・人名用漢字表にあわせて11字の字形変更.

また，JIS区点コードの115-01から119-12までは，IBM拡張コードとして，ローマ字，特殊記号，IBM選定漢字などが割り当てられています.

◎特殊文字に関しては，直接対応することができないので，外字コードとして登録するしか方法はありません．また，罫線に関しては，98罫線からJIS罫線に変更します．2バイト半角はロジックを変更して，1バイト半角文字を使用してください.

0.5 参考文献

DOS/Vでのプログラミングを行うときの参考となる資料は，どのレベルのプログラムを行うのかによって，大幅に異なります．まず，C言語だけで書けるようなプログラムでしたら，通常のPC-9801用のC言語リファレンスでまったく問題ありません．グラフィック関連もほとんど同等の関数がありますし，BIOSまわりさえ使用しなければ，ほとんどそのまま使用できます.

つぎに，MS-DOSレベルに関しても，とくに大きな違いはありません．私がよく参考にしているのは，以下の文献です.

『MS-DOSエンサイクロペディアVOL.1　システム解説編』，マイクロソフトプレス編，エー・ピー・ラボ訳，アスキー，1989

『IBM DOSバージョンJ5.0/V ユーザーズ・ガイド』，N:SC18-2488

『IBM DOSバージョンJ5.0,J5.0/V 技術解説書』，N:SC18-2490

また，BIOSレベルのプログラミングを行うのでしたら，以下の文献が参考になると思います.

『IBM DOSバージョンJ5.0/V BIOSインターフェース技術解説書』，N:SC18-2489

『IBM PS/55マウス・ドライバー　ユーザーズ・ガイド（DOS/V用）』，N:SH18-2314

『IBM PS/55キーボード技術解説書』，N:SA18-7472

『IBM PS/55ディスプレー・アダプター技術解説書』，N:SA18-7471

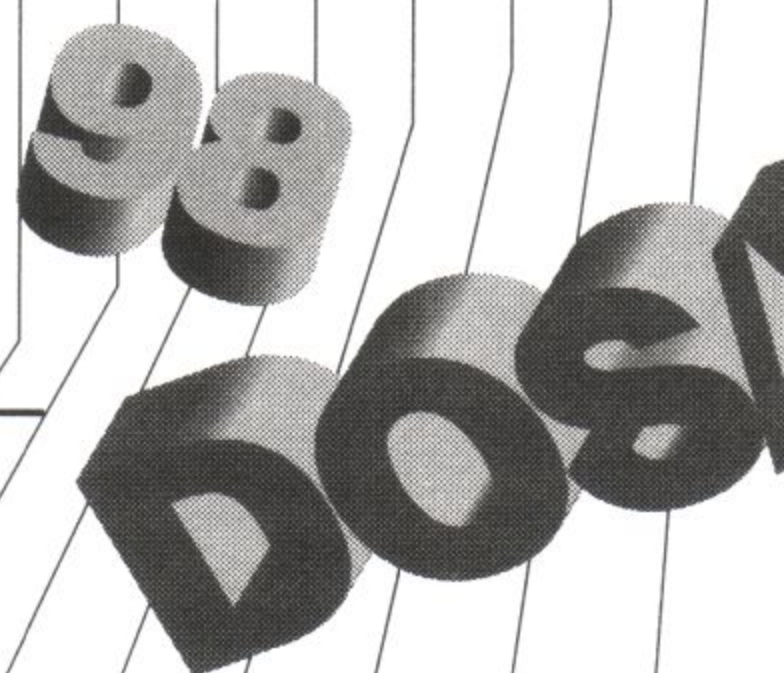
『IBM PS/55Z 10T技術解説書』，N:SA18-7345

『IBM PS/55note N23SX技術解説書』，N:SA18-7378

『THE IBM PC&PS/2プログラマーズガイド』，P.Norton, R.Wilton著，SE編集部訳，翔泳社，1989
『PC&PS/2ビデオシステムプログラマーズガイド』，R.Wilton著，SE編集部訳，翔泳社，1989

IBM関係のマニュアルは，IBM特約店または紀ノ国屋書店IBM箱崎店（TEL.03-3808-1645）で扱っています．

第1章

98からDOS/Vの世界へ

DOS/Vでのプログラミングとは

この章では，DOS/V特有のプログラミングを行う場合に，どのようなレベルでコーディングを行ったらよいのか，また，基本的なDOS/Vの環境の把握・設定方法に関して解説します．

1-1 プログラミングのレベルを決める

DOS/V環境でプログラミングをする場合，どのレベルでコーディングするかが非常に重要となってきます．たとえば，文字出力ひとつをとっても，以下のように判断しなければならないことがいろいろあります．通常，DOS/V環境で画面出力を行う場合，以下の4種類の方法が考えられます．

DOSの標準出力だけを使用

この場合，通常は文字出力だけであり，基本的に文字属性を変更することはできません．ただし，エスケープシーケンス文字を使用することにより，文字属性を変更できますが，以下の注意点があります．

PC-9801の場合，標準でもエスケープシーケンスが使用できますが，IBM PC系の場合通常は，ANSI.SYSまたは同等の機能をもつデバイスドライバを組み込まないと，使用することはできません．したがって，画面を細かく制御しようという場合，ANSI.SYSが組み込まれているかどうかがわからないため，通常はあまり使用されていません（組み込みの判定方法は後述します）．ただし，DOS5.0から，デバイスドライバをUMBにロードすることができるようになったため，最近はANSI.SYSをロードしている人も多いと思われます．しかし，エスケープシーケンス文字を使用する場合には，必ずドキュメントに「ANSI.SYSまたは同等の機能をもつデバイスドライバをロードしておいてください．」と記載しておくのが親切でしょう．

また，この場合はPC-9801と比較すると，一部のエスケープシーケンス文字がサポートされていませんが，基本的には変わるところはありません．しいていえば，画面サイズに考慮するくらいでしょう．とくに入出力リダイレクション（画面出力をファイルに落とすとかの指定）ができなくてもよい場合は，以下のINT 29hを使用するほうが，高速になります．

INT 29hを使用

INT 29hは，DOSの文字出力を高速化するインターフェースですが，IBM PC系の場合は，内部ではビデオBIOS機能（AH=0Eh：テレタイプ式文字の書き込み）が呼び出されているだけです．また，IBM PC系ではエスケープシーケンス文字はビデオBIOS内部ではなく，ANSI.SYSが処理しているので，ANSI.SYSが組み込まれている場合には，ANSI.SYS

がINT29hをフックしてエスケープシーケンス文字の処理を行っています．ANSI.SYSが組み込まれていない場合には，単純に文字出力を行っています．エスケープシーケンス文字のサポート範囲に注意すれば，DOS自身のオーバーヘッドもなく，DOS汎用のプログラムを作成することができます．

ビデオBIOSを使用

通常，画面制御はこれだけで行えてしまいます．PC-9801と違い，BIOSがいろいろな機能をもっているため，わざわざ，VRAM直書きやエスケープシーケンスを使用しなくてもいろいろな画面出力が可能となっています．当然，前述のANSI.SYSも不要です．多くのDOS/V対応ソフトウェアも，この機能を使用して画面制御しており，BIOSの使用方法を理解するのが，DOS/Vプログラミングの理解にいちばん効果的です．また，仮想VRAMへの直書きも可能ですが，基本的にはビデオBIOSを使用するため，この範囲にいれておきます．

また，ビデオBIOSではありませんが，各種SVGAプログラミングを共通化させるために，VESAと呼ばれる呼び出しインターフェースをサポートするようになってきています．

直接，VGAなどのハードウェアを操作

高速ゲームなどの場合，VGAレジスタを直接操作したり，ビデオボードごとに直接CRTCチップのレジスタを操作したりする場合があります．しかしこの場合，各チップのレジスタ間に互換性がない場合が多いので，環境設定などでビデオボードの種類を入力させるのが普通です．しかしながら，個人でプログラムを作成するのでしたら，ビデオボードの進化にはついていけないでしょうから，なるべくビデオボードなどの変化の激しいハードウェアを直接操作することは避けるべきです．

画面まわりではありませんが，直接ハードウェアを操作しなければいけないものの例外としてはRS-232Cまわりがあげられます．これは，IBM BIOSのRS-232Cまわりは必要な機能がほとんどなく，RS-232Cを操作するプログラムを作成する場合に限っては，ハードウェアを直接操作するコーディングが行われています.それでも通用してしまうのは，RS-232CまわりはすべてINS8250Bと上位互換のシリアルコントローラが使用されていますので，機種間の互換性が高いためだと思われます．筆者自身も通信プログラムを書いたことがありますが，機種により動かないとの報告は一件もありませんでした（バグの報告は数多くありましたが）．

このように，DOS/V環境でプログラムを作成する場合，DOS，BIOS，ハードウェアといったいろいろなレベルでコーディングをすることができます．DOSレベルのコーディングに関しては，とくにDOS/Vだからといって特殊なことはありません．PC-9801と比較した場合，日本語モードと英語モードを切り替えることができますので，そのぶん多国語言

語対応になっているといった点が異なっている程度でしょう．BIOSを使用するためには，タイプ10h～1Ahのソフトウェア割り込みを実行することによりによって，DOSを経由しないで直接，システム操作を行うことができます．

IBM PC系の場合，BIOSの上位互換性を固く守っているため，長期にわたって安定した互換性が保証されています．ただし，IBM　PC系から派生した機種（たとえば，AX系やJ3100系など）では独自に拡張した部分がありますので，IBM PC系で汎用に使用する場合は，共通のBIOSだけを使用するか，または，後述の動作環境の判定方法を使用して，一部の共通でないBIOS機能部分に関しては機種コードを参照して，それぞれ別途にプログラミングする必要性があります．

BIOSを使用するためには，一般的にAHに機能コードを（必要なら，その他のレジスタにパラメータを）セットし，ソフトウェア割り込み（INT命令）でBIOS機能を呼び出します．そのため，アセンブラに関する多少の知識が必要です．最近は，C言語からもBIOS機能を呼び出す関数群が標準でサポートされるようになってきていますので，簡単にプログラミングできるようになっています．ただし，本書では機能の詳細がわかりやすいように，MASM　6.0の構造化アセンブラ機能を使用してサンプルプログラムを作成しています．このBIOS機能を使用することにより，ハードウェア依存性をなくし，どのAT互換機でも使用でき，かつ，気のきいたプログラムが作りやすくなるので，このレベルでコーディングされるのがよいと思います．

1-2　動作環境の判定方法

DOS/V環境下で動作するプログラムを作成する場合，もうひとつ考えておかなくてはならないことがあります．それは，プログラムの動作環境の問題です．プログラムの動作環境には，PC-DOS，DOS/V日本語モード，DOS/V英語モードなどあり，基本的にはこれらの動作環境を把握し，その動作環境に基づいたプログラミングを行う必要があります．または，自分でプログラムの動作環境に合わせる手段を講じるべきでしょう．そこでまずはじめに，これらの動作環境を得るための方法をサンプルプログラムをもとに解説します．このプログラム（ismachin.asm）は私が使用している各種機種・DOS判定ルーチンです．

このプログラムでは，DOS/V日本語モードとUS PCモードの判定のほかに，PS/55用のIBM日本語DOS・DR DOS/V，J3100日本語モード，AX日本語モードおよびPC-9801の判定を行っています．これらのDOS/V以外の機種に関して，本書内では詳細に触れませんが，DOS/Vと同様に，PS/55日本語DOS，DR DOS/V，J3100，AXとも，基本はIBM PC/ATから派生しており，限定された範囲内でコーディングを行えば，共通なプログラムを作

リスト1.1　動作機種判定ISMACHIN.ASM

```
include         std.inc

JDOS            EQU     0                               ; PS/55日本語DOS
PCDOS           EQU     1                               ; PC DOS
DOSVJ           EQU     2                               ; DOS/V日本語ﾓｰﾄﾞ
DOSVE           EQU     3                               ; DOS/V英語ﾓｰﾄﾞ
AXJ             EQU     4                               ; AX日本語ﾓｰﾄﾞ
AXE             EQU     5                               ; AX英語ﾓｰﾄﾞ
J3100J          EQU     6                               ; J3100日本語ﾓｰﾄﾞ
J3100E          EQU     7                               ; J3100英語ﾓｰﾄﾞ
DRDOSJ          EQU     8                               ; DR-DOS日本語ﾓｰﾄﾞ
DRDOSE          EQU     9                               ; DR-DOS英語ﾓｰﾄﾞ
PC9801          EQU     10                              ; PC-9801
PC9801H         EQU     11                              ; PC-9801 Hireso

                .code

;*****************************************************************************
;*                                                                           *
;*          動作環境のチェック                                               *
;*          int         isMachine(void);                                     *
;*          戻り値：    0 = PS/55 日本語DOS                                  *
;*                      1 =       英語モード                                 *
;*                      2 = DOS/V 日本語モード                               *
;*                      3 = DOS/V 英語モード                                 *
;*                      4 = AX    日本語モード                               *
;*                      5 = AX    英語モード                                 *
;*                      6 = J3100 日本語モード                               *
;*                      7 = J3100 英語モード                                 *
;*                      8 = DRDOS 日本語モード                               *
;*                      9 = DRDOS 英語モード                                 *
;*                     10 = PC-9801 ノーマル                                 *
;*                     11 = PC-9801 ハイレゾ                                 *
;*                                                                           *
;*****************************************************************************
RevAddress      dd      00FE0000h                       ;東芝 日英DOS 5.0 判定ｱﾄﾞﾚｽ
TDos            byte    "BREVAA"                        ;判定文字列
TDos_BYTE       =       offset $ - offset TDos          ;判定文字列長

isMachine       proc    uses ds es si di
                VIDEO   0Fh                             ;画面ﾓｰﾄﾞの取得
                .if ah == 0fh                           ; PC9801系の場合
                    mov     dl,PC9801                   ; PC9801 ﾉｰﾏﾙ
                    xor     ax,ax                       ; ﾊｲﾚｿﾞ・ﾁｪｯｸ
                    mov     ds,ax
                    .if (byte ptr ds:[501h] & 08h)      ; Hireso flag
                        inc     dl                      ; PC9801 ﾊｲﾚｿﾞ
                    .endif
                .else

; DRDOS の判定

                    stc                                 ; DR DOSのﾁｪｯｸ
                    MSDOS   4412h                       ; DR DOS判定
                    .if !Carry?
                        mov     dl,DRDOSE               ; DR DOS英語
                        jmp     CheckDBCS               ;日本語の判定へ
                    .endif

; J3100 の判定

                    MSDOS   3000h                       ; DOSﾊﾞｰｼﾞｮﾝとOEM番号の取得
                    .if bh == 29h                       ;東芝OEM番号
                        jmp     Toshiba
                    .endif
                    MOVSEG  ds,cs                       ;東芝 日英DOS 5.0の判定
                    les     di,RevAddress
                    .repeat
                        mov     si,offset cs:TDos       ;比較文字列
                        mov     cx,TDos_BYTE            ;文字列長
                        push    di                      ;開始位置の保管
                        repe    cmpsb                   ; "BREVAA"文字列の比較
                        pop     di                      ;開始位置の復元
                        .if Zero?                       ;文字列が見つかった
Toshiba:                    mov     dl,J3100J           ; J-3100日本語
                            xor     ax,ax
                            mov     ds,ax
                            .if !(byte ptr ds:[4D0h] & 1)   ;英語ﾓｰﾄﾞの判定
                                inc     dl              ; J3100英語
                            .endif
                            jmp         isMachineExit
                        .endif
                        inc     di                      ; 一文字分進める
                    .until di == 10h                    ; 00FEh:0h～10hまでをチェック

; AX の判定

                    xor     ax,ax
                    mov     ds,ax
                    mov     si,04e0h                    ; 40h:0e0h から 8ﾊﾞｲﾄをﾁｪｯｸ
                    mov     cx,4
                    .repeat
                        lodsw
                        .if ax != 0                     ;非 0 なら AX
                            xor     bx,bx
                            VIDEO   5001h               ; AX言語ﾓｰﾄﾞ・ﾁｪｯｸ
                            mov     dl,AXJ              ; AX日本語
                            .if bx == 1h
                                inc     dl              ; AX英語
                            .endif
                            jmp     isMachineExit
                        .endif
                    .untilcxz

; PC-DOS / DOS/V / PS/55日本語DOS の判定

                    mov     ax,4900h
                    int     15h                         ; BIOSﾀｲﾌﾟの取得
                    .if carry? || bl != 0               ;日本語DOS or PC-DOS
                        xor     ax,ax
                        mov     ds,ax
                        mov     ax,ds:[7Dh*4]           ; INT 7Dhの使用のﾁｪｯｸ
                        or      ax,ds:[7Dh*4+2]
                        mov     dl,PCDOS                ; PC-DOS
                        .if ax != 0
                            dec     dl                  ; JDOS日本語ﾓｰﾄﾞ
                        .endif
                    .else                               ; DOS/V
                        mov     dl,DOSVE                ; DOS/V英語ﾓｰﾄﾞ
CheckDBCS:              MSDOS   6300h                   ; DBCSﾍﾞｸﾀｰの取得
                        mov     ax,ds:[si]
                        .if ax != 0
                            dec     dl                  ; DOS/V or DR DOS日本語ﾓｰﾄﾞ
                        .endif
                    .endif
                .endif
isMachineExit:
                mov     al,dl                           ; ax =機種コード
                xor     ah,ah
                ret
isMachine       endp

                end
```

成することができるからです．また，日本国内ではかなりのシェアをもつPC-9801に関しても，機種判定は行えるようにしています．

リスト1.1のプログラムは，以下の要件から作成されています．

IBM PC系の場合，AH=0Fh,INT 10hは現在のビデオ状況の取得機能になってますが，PC-9801の場合は，プリンタおよびNDP割り込みになっていて，通常は呼び出したときのレジスタはそのまま保存されます．したがって，呼び出し後，AHの値が0Fhのままであれば（IBM PC系の場合は表示桁数を示しますので，桁数が15桁のIBM PCがあれば別ですが），機種はPC-9801と判定できます．また，PC-9801のBIOSワークエリアの値を判定し，ノーマルモードかハイレゾリューションモードかの判定も行っています．

2番目は，DR DOSの判定です．DR DOSを判定するために，以下の呼び出しを使用します．また，DBCSテーブルを見て，日本語モードか英語モードかを判定します．

INT 21：DR DOS 5.0<DOSタイプの判定>

AX = 4412h

　CFをオンにセットする

[戻り値]

DR DOSでない場合，CFはオンのまま

DR DOSのときは，CFはクリアされる

　DX = AX = バージョン番号

　　1065h = DR DOS 5.0

　　1067h = DR DOS 6.0

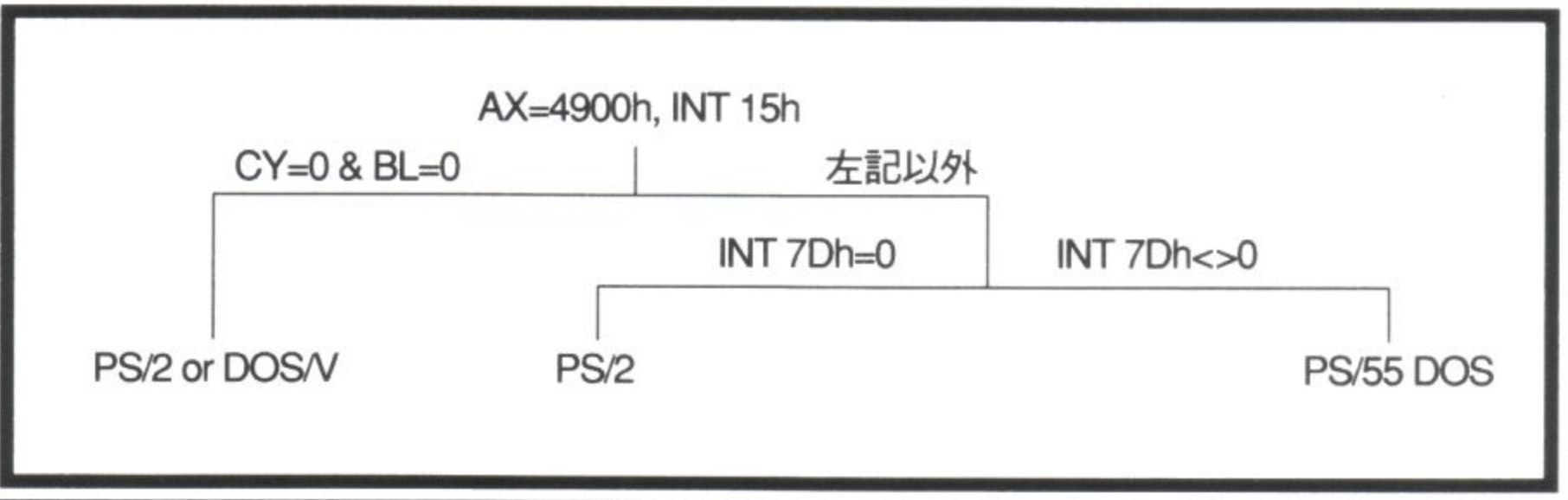
つぎに，MS-DOSのOEM番号より，J3100（Ver 3.1）を判定します．これは，J3100の日本語DOSはOEM番号として29hを返すからです．しかし，日英DOS5.0の場合は，OEM番号を返しませんので，内部にもっているリビジョン番号の文字列で判定しています．また，非公開情報ですが，BIOSワークエリアの値により，日本語・英語モードの判定を行います．

AXパソコンの場合は，セグメント40hからはじまるBIOSワークエリアのうち，E0hからEFhの部分をAX系の処理のために使用すること（本来のIBM PC系では使用しないのですが）を利用して，その先頭8バイト部分に0以外の値が入っていれば，AX系と判断しています．ただし，この方法は最近のAX VGA/SやVGA/Hに関しては判定できません．また，日本語・英語の判定は，AX BIOS割り込みを使用して判定しています．

最後にBIOSタイプの取得（AX=4900h,INT 15h）を使用して残りのBIOSモードを判定しています．アプリケーションが実行されるシステムBIOSには2種類あります．PS/2 BIOSまたはDOS/V BIOSの場合は，DBCSベクタの取得（AX=6300h,INT 21h）で日本語・英語の判定を，PS/55日本語DOSの場合は，INT 7Dhベクタの使用の有無を利用して，日本語DOSとUS PCモードの判定を行っています．

この関数を応用したものとして，添付ディスク中にモード判定プログラムのismode.com（ソースコードはismode.asm）を添付しておきます．

図1.1

DOS/V BIOSモードの判定ロジック

1-3 英語モードと日本語モードの切り替え

DOS/Vの本体は，IBMBIO.COM（ハードウェアに依存した基本的な入出力を担当する部分）とIBMDOS.COM（MS-DOSとしての処理を行う部分）の2つの隠しファイルとCOMMAND.COMおよび各種DOSユーティリティから構成されています．

これらのモジュールは，基本的には英語版のPC-DOS（MS-DOS）となんら変わることがなく，英語版PC-DOS 5.0にDOS J5.0/VやSuper Driversの日本語ドライバをのせて，DOS/V環境を構築することも可能です．

実際には，DOS/Vの各モジュールは日本語環境でより便利に使用できるように，漢字ファイル名の処理や英語メッセージと日本語メッセージの二重化などの漢字処理機能の拡張が行われていますが，あくまでも表面的な変更だけにとどまり，大きな機能変更などは一切ありません．

DOS/Vでは，この点をうまく利用して，英語モードと日本語モードをダイナミックに切り替えることが可能です．そのために，CHEVおよびSWICTHコマンドの2種類が用意されています．

CHEVとSWITCHコマンドの基本的な違いは，メモリ容量です．すなわち，CHEVコマンドによる環境切り替えは，$FONT.SYSや$DISP.SYSなどの日本語環境特有のデバイスドライバをそのままメモリ上に残して，日本語処理の実行／非実行を切り替えて，日本語環境と英語環境を使い分けています．そのため，高速に日本語環境と英語環境の行き来ができ，通常のアプリケーションはほとんどこのモード切り替えで使用できます．しかし，日本語モードのデバイスドライバや常駐プログラムがそのまま残っており，メモリを圧迫している点，および，それらのプログラムが英語環境に対応していない場合もあるので注意は必要です．

それに対して，SWITCHコマンドの場合は，完全に，CONFIG.SYSとAUTOEXEC.BATをそれぞれのモードのものに変更してから，リブートを行うため，ほぼ完全な英語版のPC-DOSとして使用できます．

DOS/Vで日本語環境と英語環境を判定する場合，以下のDBCSベクタテーブルとコードページの2か所をチェックしています．

図1.2
日本語環境と英語環境の切り替え

DBCSベクタテーブル

DBCSベクタテーブルとは，漢字のような2バイト文字セット（DBCS=Double Bytes Character Set）を判定するために使用される，第1バイト目の範囲を示しているテーブルです．このDBCSベクタテーブルを見つけるためには，DOSのシステムコール（AX=6300h DBCSベクタ情報の取得）を使用します．

```
MSDOS   6300h<DBCSベクタテーブルのアドレスを取得する>
[戻り値]
carry =         1  エラー（AX = エラーコード）
                0  正常終了
DS:SI           DBCSベクタテーブルの先頭アドレス
```

DBCSベクタテーブルは日本語環境のときは，以下のレイアウトになっています．

```
DS:SI-2 word  0006h       ; DBCSテーブルの大きさ
DS:SI   byte  81h, 9Fh    ; DBCSベクタ1
DS:SI+2 byte  0E0h, 0FCh  ; DBCSベクタ2
DS:SI+4 byte  00h, 00h    ; DBCSベクタの終わり
```

また，英語環境のときは，

```
DS:SI-2 word  0000h       ; DBCSテーブルの大きさ
DS:SI   byte  00h, 00h    ; DBCSベクタの終わり
DS:SI+2 byte  00h, 00h    ;
DS:SI+2 byte  00h, 00h    ;
```

となっています．DOSのシステムコールをトレースしてみると，DBCSベクタ情報の取得機能を使用して，日本語モードか英語モードかを判定しています．とくに後述するビデオモードを変更する際には，このDBCSベクタテーブルの値が0であれば，英語環境と判定して，英語のビデオモードに設定してくれます．逆に言えば，DBCSベクタテーブルを書き換えても，ビデオモードを変更しないと画面の状況はそのままですので，英語環境の画面に間違えて日本語メッセージを表示するような状態になる場合や，日本語環境なのにテキストVRAMに直接書き込んでしまい文字が表示されないといった状態が生じてしまいますので，必ず，DBCSベクタテーブルを書き換えた場合にはビデオモードも再度設定し直してください．

日本語環境と英語環境を切り替えない場合でも，自分でプログラムを作成する場合は，このシステムコールを使用して，環境が日本語なのか英語なのかを判定して，メッセージを切り替えたり，動作できない環境であれば，「環境を切り替えてください．」のようなメッセージ（当然，英語環境でしたら英語で同じメッセージを出しましょう）を出力するようにすればよいわけです．

もう一歩進んだプログラムの場合には，リスト1.2で示すような関数を使用して，強制的に日本語環境と英語環境を切り替えてしまうことも可能です．

リスト1.2　　**日本語環境と英語環境の切り替え関数**

```
include         std.inc

                .code

;****************************************************************************
;*                                                                          *
;*      日本語環境と英語環境の切り替え                                       *
;*      void        ChgEnvMode(int newLangMode, int newVideoMode);          *
;*                                                                          *
;****************************************************************************
ChgEnvMode      proc    uses bx cx dx ds es si,¥
                                newLangMode:word, newVideoMode:word
                VIDEO   0Fh                     ;現在のビデオモードの取得
                mov     cl, al                  ; clに保存

                PUSHM   <ds,si>
                MSDOS   6300h                   ; DBCSﾍﾞｸﾀ の取得
                mov     bx, ds:[si]             ; 現在のDBCSﾍﾞｸﾀの値
                                                ; == 0 なら英語モード
                                                ; != 0 なら日本語モード
                POPM    <si,ds>

                mov     ax, newLangMode
                mov     dx, newVideoMode

                .if (dl != cl) || (ax != bx)    ;ビデオモードが異なるか
                                                ;言語が異なる場合
                    .if ax == 0                 ;英語モードにする
                        MSDOS   6601h           ;グローバルコードページの取得
                        .if !carry?
                            mov     bx, 437     ; US
                            MSDOS   6602h       ;グローバルコードページの変更
                        .endif
                        MSDOS   6300h           ; DBCSﾍﾞｸﾀの取得
                        assume  si:PWORD
                        mov     ds:[si-2], 0    ; DBCSﾍﾞｸﾀを消去する
                        mov     ds:[si],   0
                        mov     ds:[si+2], 0
                        mov     ds:[si+4], 0
                        assume  si:nothing
                    .else                       ;日本語モードにする
                        MSDOS   6601h           ;グローバルコードページの取得
                        .if !carry?
                            mov     bx, 932     ; JAPAN
                            MSDOS   6602h       ;グローバルコードページの変更
                        .endif
                        MSDOS   6300h           ; DBCSﾍﾞｸﾀの取得
                        assume  si:PWORD
                        mov     ds:[si-2], 6    ; DBCSﾍﾞｸﾀの設定
                        mov     ds:[si],   9f81h
                        mov     ds:[si+2], 0fce0h
                        mov     ds:[si+4], 0000h
                        assume  si:nothing
                    .endif

                    mov     al, dl              ;新しいビデオモード
                    VIDEO   0                   ;ビデオモードの設定
                .endif
                ret
ChgEnvMode      endp

                end
```

コードページ

DOSで表示・印刷できる1バイト文字セット（SBCS＝Single Byte Character Set）のことをコードページと呼びます．これは，使用される国によって言語が異なるためで，コードページを切り替えることによって，他国語の文字を入力，表示，印刷することができます．

DOS/Vでサポートされているコードページには，437（US）と932（日本）の2種類があります．このコードページを操作するのはDOSのシステムコール（AH=66hグローバルコードページの取得と設定）を使用します．

MSDOS　6601h＜グローバルコードページの取得＞
[戻り値]
carry =　　1　エラー（AX = エラーコード）
　　　　　　0　正常終了
BX =　　　　グローバルコードページ
DX =　　　　システムコードページ

BX =　　　　新しいグローバルコードページ
MSDOS　6602h＜グローバルコードページの設定＞
[戻り値]
carry =　　1　エラー（AX = エラーコード）
　　　　　　0　正常終了

ただし，このシステムコールを使用する場合には，NLSFUNC.EXEが組み込まれていなければなりません．NLSFUNC.EXEは英語環境でのみ有効のため，通常は，日本語環境では使用されませんが，なかにはチェックしているプログラムがあるかもしれませんので，

一応対応して，グローバルコードページの取得機能が正常に実行された場合に，グローバルコードページを切り替えるようにしています．

第2章

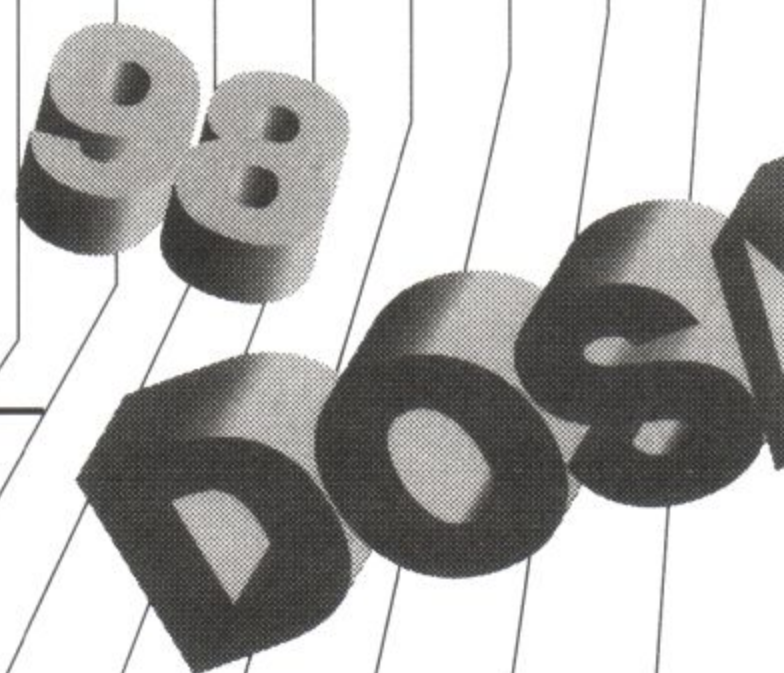
98からDOS/Vの世界へ

98 DOS/V

画面制御編

　この章では，DOS/Vとしてもっともユニークである画面制御に関して解説します．とくに，テキスト画面（とくにV-Textまわり）を中心に取り上げます．

　V-Textに対応した画面・キーボードまわりのプログラムパーツを作成しながら，DOS/Vの基本機能を理解していただきたいと思います．まず，単純なDOS対応プログラムを作成するために，エスケープシーケンス文字の処理を取り上げます．前述したように，DOS/V環境では，ANSI.SYSを使用しないとエスケープシーケンス文字が使用できないため，ANSI.SYSが組み込まれたかどうかを確認する関数を作成します．また，使用できるエスケープシーケンス文字に関しても解説します．

　つぎに，ビデオBIOSを使用して，カーソルの制御，画面上の文字の読み取り方法，書き込み方法や仮想VRAMに直接アクセスする関数，および，パレット設定やフォント登録に関しての関数群を作成します．

2-1 基礎知識

DOS/Vで画面処理を行うプログラムを作成するうえで，どうしても知っておかなくてはならないことがいくつかあります．ひとつは，DOS/Vそのものの日本語表示のしくみです．DOS/Vはソフトウェアによって画面に日本語を表示しているわけですから，プログラムを作成する場合にはそのしくみを知っておく必要があります．もうひとつは，「ビデオモード」についてです．IBM PCでは，PC-9801の場合のように画面を80桁×25行の固定で考える必要はありません．PC-9801では，ノーマルモードでは80桁×25行または80桁×20行をエスケープシーケンスによって切り替えられますが，DOS/VではビデオBIOSを使用することにより，テキスト画面とグラフィック画面の切り替え，表示文字数や画面サイズを切り替えることができます．また，DOS/V Super DriversやIBM DOS/V Extension V1.00を使用することにより，より高品位で，より高密度（場合によっては低密度）なビデオ環境を使用できるようになっています．

まず最初に理解していただきたいのは，IBM PC系の場合，PC-9801系のように，テキスト画面とグラフィック画面との重ね合せはできないということです．基本的には，ビデオモードによりどちらかひとつを設定するということになります．ただし，VGA自身が16色=4プレーンをもっていますので，市販ソフトウェアのなかにはVGAレジスタを直接制御することにより，文字表示に2プレーン，グラフィック表示に2プレーンといった使い方をしているものもあります．また，DOS/Vテキストモードの画面は実際はグラフィック画面を使用していますから，きちんとVGAレジスタの処理をすれば，テキスト画面中に16色までのグラフィックやイメージの表示を行うこともできます．しかし，現実にはビデオチップにより動作が異なる場合がありますので，非常に難しいと思っておいたほうがよいでしょう．

図2.1に示すように，AT互換機には128KBの表示用VRAMが用意されています．この領域は実際にはメインボードではなく，装備するビデオボード上にあります．一般的な使用

図2.1

AT互換機のVRAM構成

セグメント	
C000〜B800	カラーテキストVRAM (32KB)
B800〜B000	モノクロテキストVRAM (32KB)
B000〜A000	グラフィックVRAM (64KB)

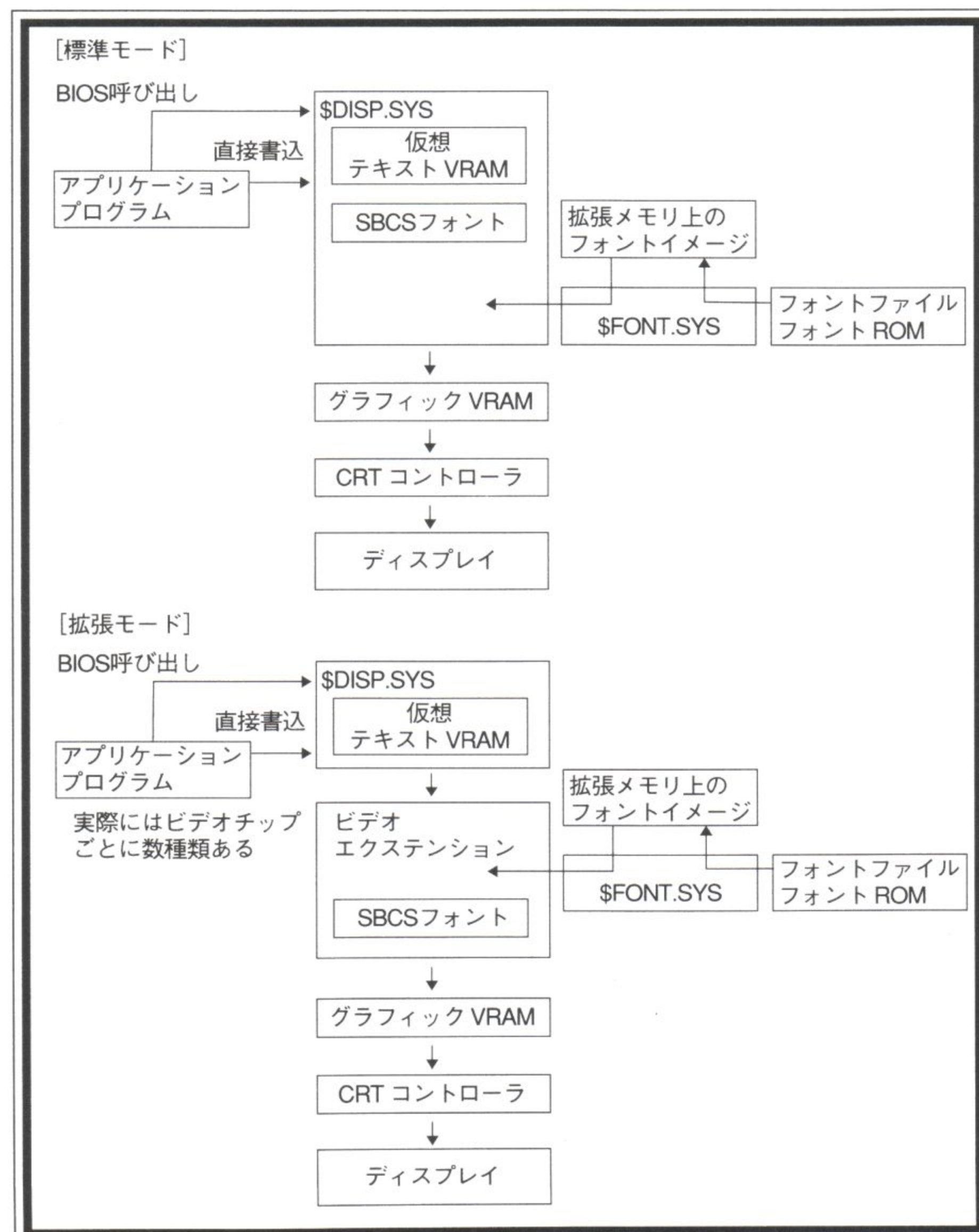
注）モノクロテキストVRAMは，日本語モードでは使用されることはなく，UMB領域または，WindowsやV-Textで高解像度表示(1024×768）時に，拡張グラフィックVRAMとして使用されることが多い．

形態は図2.1に書いてあるとおりですが，実際にはビデオモードによってさまざまに使用形態は変化します．このため，PC-9801のように固定的ではなく，柔軟にグラフィック機能を向上させることが可能となっています．

$FONT.SYSと$DISP.SYSのはたらき

DOS/Vで日本語環境を実現するために重要なのが，$FONT.SYSと$DISP.SYSの2つのデバイスドライバです．DOS/Vの基本的なしくみは，英語版のPC(MS)-DOSに日本語のフォントの展開および表示を行うドライバを付け加えて，BIOSを日本語拡張するというものです（当然，2バイト文字のハンドリングやメッセージの日本語化なども行われましたが……）．

この日本語ドライバとして，$FONT.SYSは拡張メモリ上に日本語フォントを展開し，プログラムからの要求にしたがい，フォントの読み出しや登録などの管理を行います．$DISP.SYSは，$FONT.SYSより得られたフォントをグラフィック画面に展開し，日本語の表示処理を行っています．このため，ビデオBIOSを使用したプログラムは特別なハードウェアなしに，日本語を表示できるようになっています．後述するV-Textドライバも，基

図2.2

DOS/V日本語表示処理のしくみ

本的に$FONT.SYS，$DISP.SYSとしくみは同じです．

$DISP.SYSはROM BIOSにもっているビデオサブシステム（画面処理を担当している）を拡張する日本語環境下の画面表示ドライバで，日本語モード時にフォントイメージをフォントドライバ（$FONT.SYS）から取得して画面に表示しています．

AT互換機では，ハードウェア的に日本語表示機能はもっていないため，日本語モードの場合は，VGA，SVGA，XGAの16色グラフィック画面に表示を行っています．そのため，実テキストバッファ（VRAM）は存在していません．ドライバだけで構成されているため，自由度が高く，最近では，V-Text（以前はハイテキストと呼ばれていました）と呼ばれる高品位・高密度テキスト画面が標準となりつつあります．

V-Text（Variable-Text）

DOS/Vの特徴としては，画面まわりの自由度の高さがあげられます．とくに，V-Textと呼ばれる高品位・高密度の環境があるため，DOSの世界でもいままでのように，80桁×25行に固定して考える必要はありません．

V-Textドライバとは，DOS/Vのテキスト画面を拡張するドライバで，これを使用すると各種多様の画面解像度（100×31，100×37，128×48，132×50など）のテキスト画面の表示，切り替えができるようになります．また，縦書きや12/24ドット漢字への切り替えなども可能になっていて，通常のVGA画面（とくに，ノートブックパソコンなど）でも，V-Textが可能となっています．

これらの高解像度画面を使用するメリットとは何なのでしょうか．基本的には高品位性と情報量です．いままでの24ドット対応のパソコンは，非常に高価であり，個人で買うには辛いものがありました．V-Textを使用すれば，個人クラスのパソコンで，24ドットの高品位画面を見ることができます．情報量に関しては，Windows上では，昨今高解像度の画面表示ができることをセールスポイントとしたハードウェアが登場しつつありますが，V-Textとは，いわばDOSの世界での高解像度画面だと思えばよいと思います．

たとえば，フリーソフトウェアのFDやFILMTNを知っているでしょうか．どちらもファイルを操作するためのシェルですが，通常の80桁×25行画面表示ですと，3段表示の場合は，51ファイル表示で，ファイル名とバイト数しかわかりません．しかし，100桁×37行モードならば84ファイルを表示し，かつファイル名＋バイト数＋日付まで表示することができます．また，パソコン通信でチャットやRTをサポートするCA（チャットアダプタって知らないかな）でも，横が100桁以上なら，ハンドルウィンドウを常駐表示するようになっています．

さらにいえば，市販ソフトでも，「VJE-PEN」，「知子の情報」，「弥生の会計」など，V-Textに対応しているものが順次増えてきています．ワープロで横が全角40文字しか画面に入らずに困った人はいないでしょうか？　A4縦書きの場合，文字間隔にもよりますが，横に全角43文字前後がちょうどよいはずです．V-Textに対応することにより，これまで画面上でいちいち横スクロールしていたものが，簡単に全桁を見通せるようになり，パ

フォーマンス的にもよいものとなり，操作性が向上するはずです．

弥生の会計は私自身見たことはありませんが，話によれば，会計伝票が一画面で表示できるようになり，わかりやすくなったとのことです．

代表的なV-Textドライバには，以下のようなものがあります．

IBM	DOS/V Extension V1.0
C.F.Computing	Super Drivers
フリーソフトウェア	DISPV.EXE
フリーソフトウェア	DISPS3.EXE
フリーソフトウェア	$DISPH.SYS

しかしながら，このV-Textもすべてがいいことづくめではありません．V-Textを利用するためのソフトウェアのほとんどが，フリーソフトウェアで固められているということから，NIFTY-Surveや日経MIXに加入している方なら簡単に手に入れられるのですが，それ以外の方法ではなかなか入手できません．また，ビジネスで使用する場合，不安感が残ることなどの問題があげられます．

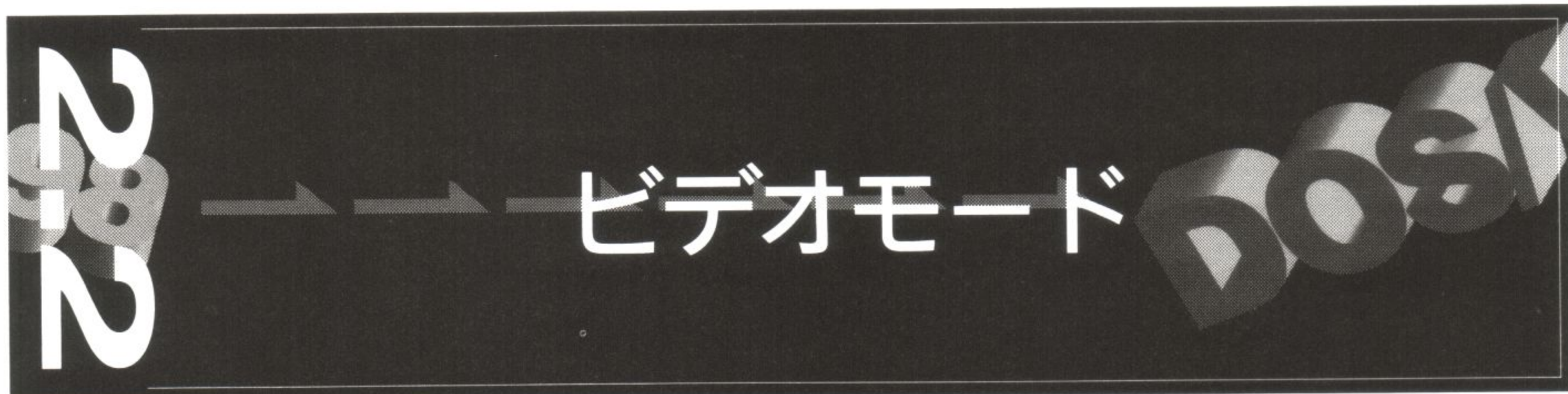
しかし，この点に関しても事態は解決に向かいつつあります．C.F.Computingが監修したV-Textドライバソフト付き書籍（『DOS/Vスーパードライバーズ』，9,800円）が，ソフトバンクから販売されたのです．また，驚くべきことに，御本家の日本IBMでもV-Textを標準として認め，93年3月末にIBM DOS/V Extension Ver.1.0（7,000円）として出荷されました．また，DR DOSでも，V-Textに対応するようになりますので，そろそろ，このV-Textも規格として定着するものと考えていいと思います．今後，V-Text対応のソフトウェアが増えてくることが期待されます．

したがって，DOS/V環境でプログラミングをする場合，80桁×25行の画面で動作することが最低条件ですが，なるべく画面の解像度をチェックして，80桁×25行以上ならば，それに対応できるようにプログラミングするべきだと思います．とくに縦方向の拡張は，行数を増減させるだけで済みますので，対応しやすいと思われます（横方向の拡張は，何を表示させるかが問題になってくるので，そう簡単には対応できないかもしれませんが）．しかし，どちらにしろ画面の範囲を広げたり，ほかの情報を表示するようにすれば，より多くの情報を画面に表示でき，より使いやすいプログラムができあがるはずです．

また，V-Textドライバには画面を高密度にするだけではなく，高品位にするという特徴もあります．24ドットフォントを使用すれば，V-Text対応プログラムでなくても，一部機能(フォントの登録など)を除いては,そのまま高品位の画面を表示させることができます．

表2.1　代表的な解像度

テキストモード	桁数	行数	付記
低解像度（LowText）	80	12	V-Textドライバ
標　準	80	25	DOS/V標準
	80	30	
高解像度（HiText）	100	31	V-Textドライバが別途必要．
	100	37	
	128	44	

これらの画面解像度をプログラムから操作するには，ビデオBIOSを使用して，ビデオモードを切り替えるだけです．たとえば，日本IBMから販売されたDOS/V Extensionでは，ビデオBIOSが拡張され，標準モードと高品位モードの切り替え（AH=12h，BL=38h，INT 10h）や高密度モードへの切り替え（AH=11h，AL=18h，INT 10h）などが可能となっています．

2-2 ビデオモード

IBM PC/ATおよびその互換機上でプログラミングするうえで知っておくべきなのが，この「ビデオモード」というものです．すでに知っているとは思いますが，IBM PCのビデオシステムは，初期のMDA（Monochrome Display Adapter）およびCGA（Color Graphics Adapter）から，EGA（Enhanced Graphics Adapter），VGA（Video Graphics Array）と発展してきました（そして，最近ではSVGA，XGAへとさらに発展し続けています）．これらのビデオサブシステムごとに，それぞれサポートしているビデオモードがあり，これらのビデオモードは固有のビデオモード番号をもっています．

解像度や色数の話を無視すると，ビデオモードは以下の3種類に大別できます．

(1) エミュレートCGAテキストモード（前景色・背景色のみ）

(2) エミュレート拡張CGAテキストモード（十下線，縦横罫線）

(3) グラフィックモード

ビデオサブシステムは，基本的に下位のものと互換性をたもちながら拡張されてきましたから，DOS/Vで前提とされているVGAでは，下位サブシステムが使用していたビデオモードに対する配慮がなされています．ただし，これらのモードはテキストモードといっても，前述のように，BIOSレベルでそのように見えるだけで，実際にはテキストモードは存在しておらず，表示ドライバがフォント展開を行い，グラフィック画面に表示しています（しかし，グラフィック画面といっても，速度的にはビット単位での処理はほとんど不要です．また，スクロール処理も，VGA用のCRTCレジスタのAPA開始アドレスを変更することで，高速に行われています．また，V-Textドライバの場合は，SVGAやXGAがもつBitBlt機能を使用したり，フォント自身をビデオバッファの空きエリアにロードするなどして，高速スクロールを実現させています）．

エミュレートCGAテキストモードは，ビデオBIOSインターフェースとしては，英語モードのCGAテキストモードと機能的に同等で，1バイト文字に対して1バイトの属性をもっています．また，2バイト文字の場合，左右別々に文字属性を付加することが可能となっています．さらに，このモードでは仮想VRAMが存在しており，より高速なアクセスが可能となっています．

これに対して，エミュレート拡張CGAテキストモードは，1バイト文字に対して，3バイトの属性をもっています．このモードでは，ユーザーがアクセスできる仮想VRAMはありませんので，後述の「文字ブロックの読み取り（AX=131xh，INT 10h），書き込み（AX=132xh，INT 10h）」を使用して，読み取り・書き込み機能を実行できます．また，縦罫線，横罫線，下線が使用できるため，よりきめ細かな画面描画が可能となっています．ただし，後者のほうが属性が多くきれいですが，仮想VRAMがないこと，および，3バイトの属性を管理しなければならず，処理も多少複雑となりますので，通常は前者のほうがよく使用されています．また，前者のほうでもいろいろ工夫すれば，結構きれいな画面を使用することができます．

また，この画面属性で指定する色は，直接，16色を指定するのではなく，64色のパレットのなかから，16色を指定することになります．グラフィックモードの場合には，背景色といった考えはなく，前景色だけが有効となり，また，最上位ビットの値で，現在のイメ

◎エミュレートCGAテキストモード

7	6	5	4	3	2	1	0

- 0：前景の青
- 1：前景の緑
- 2：前景の赤
- 3：前景の高輝度
- （ビット0〜3）前景のパレット番号
- 4：背景の青
- 5：背景の緑
- 6：背景の赤
- 7：明滅／背景色の高輝度（DOS/Vマシンでは通常は明滅をサポートしていないため，このビットは背景色の高輝度として働きます．将来的にはどうなるかわからないので，決めつけないでください）
- （ビット4〜7）背景のパレット番号

◎エミュレートCGAテキストモード

1バイト目

7	6	5	4	3	2	1	0

- 0：前景の青
- 1：前景の緑
- 2：前景の赤
- 3：前景の高輝度
- （ビット0〜3）前景のパレット番号
- 4：背景の青
- 5：背景の緑
- 6：背景の赤
- 7：明滅／背景色の高輝度（同上）
- （ビット4〜7）背景のパレット番号

2バイト目

7	6	5	4	3	2	1	0

- 0：予約済み
- 1：予約済み
- 2：横罫線
- 3：縦罫線
- 4：透過属性用に予約済み
- 5：明滅用に予約済み
- 6：反転用に予約済み
- 7：下線

3バイト目

7	6	5	4	3	2	1	0

すべて予約済み

◎グラフィックモード

7	6	5	4	3	2	1	0

- 0：前景の青
- 1：前景の緑
- 2：前景の赤
- 3：前景の高輝度
- （ビット0〜3）前景のパレット番号
- 4：無視されます
- 5：無視されます
- 6：無視されます
- 7：ビット7がオンの場合，文字イメージは現行のビデオバッファとXORされます．

図2.3 画面属性

ージとXORをされるかどうかが決定されます．

ビデオモード番号

以下に，日本語モードおよび英語モード時のビデオモードに関してまとめておきます．基本的に，IBM DOS/V，DR DOS/V，フリーソフトウェア（DISPV V1.5，DISPS3 V1.5），販売中のDOS/V Super Drivers（βリリースですので，最終的に仕様が変更されることがあります）およびIBM DOS/V Extension Ver 1.0を網羅しています．最新のV-Text関連ドライバの場合，ビデオモード70hがエミュレートCGAテキストの拡張を，71hがエミュレート拡張CGAテキストの拡張となっています．これは，IBMが発表した，IBM DOS/V Extensionでも使用されており，ビデオモード70h，71hは全世界的に標準として予約されるようになっています．

また，DOS 5.0より，最上位のビットは，ビデオモードとしてではなく，可能ならば画面を消去しないでビデオモードを切り替えるために使用されています．したがって，後述のVESA対応の場合を除いて，ビデオモードは01hから7Fhまでの範囲しか使用できません．

また，日本語モードでも，ビデオモード13h以下の日本語モードにないビデオモードを選択することもできます．ただし，この場合は画面は日本語表示ができない英語モード（システムは日本語モードのまま）になります．

表2.2 ビデオモード番号

◎日本語モード時のビデオモード番号

グラフィックモード

ビデオモード	色数	表示文字	解像度	フォントの大きさ	ページ数
11	2	80×30	640×480	16×16　8×16	1
12	16	80×30	640×480	16×16　8×16	1
72	16	80×25	640×480	16×19　8×19	1

エミュレートCGAテキストモード

ビデオモード	表示文字	解像度	フォントの大きさ	ドライバコード
03	80×25	640× 480	16×19　8×19	下記以外
03	80×25	1024× 768	24×24　12×24	4,E3
03 縦横変換	80×25	768×1024	16×19　8×19	6,E5
03	80×25	640× 200	16× 8　8× 8	EA
03	80×25	640× 400	16×16　8×16	EB
03	80×25	640× 350	16×14　8×14	EC
70	100×31	800× 600	16×19　8×19	2,9,D2,E1
70	100×37	800× 600	16×16　8×16	3,A,D3,E2
70	84×32	1024× 768	24×24　12×24	4,E3
70	132×50	800× 600	12×12　6×12	5,E4
70 縦横変換	96×53	768×1024	16×19　8×19	6,E5
70	128×40	1024× 768	16×19　8×19	7,E6
70	128×48	1024× 768	16×16　8×16	8,E7
70	80×30	640× 480	16×16　8×16	B,D1,E8
70	106×40	640× 480	12×12　6×12	C,E9
70	106×16	640× 200	12×12　6×12	EA
70	106×33	640× 400	12×12　6×12	EB
70	106×29	640× 350	16×14　8×14	EC

エミュレート拡張CGAテキストモード

ビデオモード		表示文字	解像度	フォントの大きさ		ドライバコード
71		100×31	800× 600	16×19	8×19	2,9,D2,E1
71		100×37	800× 600	16×16	8×16	3,A,D3,E2
71		84×32	1024× 768	24×24	12×24	4,E3
71		132×50	800× 600	12×12	6×12	5,E4
71	縦横変換	96×53	768×1024	16×19	8×19	6,E5
71		128×40	1024× 768	16×19	8×19	7,E6
71		128×48	1024× 768	16×16	8×16	8,E7
71		80×30	640× 480	16×16	8×16	B,D1,E8
71		106×40	640× 480	12×12	6×12	C,E9
71		106×16	640× 200	12×12	6×12	EA
71		106×33	640× 400	12×12	6×12	EB
71		106×29	640× 350	12×12	16×12	EC
73		80×25	640× 480	16×19	8×19	下記以外
73		80×25	1024× 768	24×24	12×24	4,E3
73	縦横変換	80×25	768×1024	16×19	8×19	6,E5
73		80×25	640× 200	16× 8	8× 8	EA
73		80×25	640× 400	16×16	8×16	EB
73		80×25	640× 350	16×14	8×14	EC

ドライバコード	使用ドライバ	ドライバコード	使用ドライバ
1	$DISP.SYS	9	DISPV.EXE
2	DISPS3.EXE	A	DISPVA.EXE
3	DISPS3A.EXE	B	DISPVB.EXE
4	DISPS3B.EXE	C	DISPVC.EXE
5	DISPS3C.EXE	D1	$DISP_H.SYS(/AHM=1)
6	DISPS3D.EXE	D2	$DISP_H.SYS(/AHM=2)
7	DISPS3E.EXE	D3	$DISP_H.SYS(/AHM=3)
8	DISPS3F.EXE		

DOS/V SuperDriversの場合

ドライバコード	使用ドライバ	ドライバコード	使用ドライバ
E1	S3.DRV VGA.DRV	E6	S3E.DRV WD31C.DRV
	WD31.DRV		QV.DRV 8514.DRV
E2	S3A.DRV VGAA.DRV		XGA.DRV
	WD31A.DRV	E7	S3F.DRV WD31D.DRV
E3	S3B.DRV WD31B.DRV		QVA.DRV 8514A.DRV
	QVB.DRV 8514B.DRV		XGAA.DRV
	XGAB.DRV	E8	VGAB.DRV JEGA.DRV
E4	S3C.DRV VGAD.DRV	E9	VGAC.DRV JEGAA.DRV
E5	S3D.DRV 8514C.DRV		XGAC.DRV
	XGAD.DRV	EA	CGA.DRV
		EB	J31.DRV
		EC	EGA.DRV

IBM DOS/V Extension Version1.0

ビデオモード	表示文字	解像度	フォントの大きさ		ドライバ	
03/73	80×25	1024× 768	24×30	12×30	960× 750	XGA
03/73	80×25	1040× 768	26×30	13×30	1040× 750	XGA
03/73	80×38	1024× 768	24×20	12×20	960× 760	XGA
03/73	80×42	1024× 768	18×18	9×18	720× 756	XGA
03/73	80×33	800× 600	18×18	9×18	720× 594	XGA
03/73	80×39	1280×1024	26×26	13×26	1040×1014	XGA
70/71	128×42	1024× 768	16×18	8×18	1024× 756	XGA
70/71	100×33	800× 600	16×18	8×18	800× 594	XGA
70/71	160×56	1280×1024	16×18	8×18	1280×1008	XGA
70/71	106×39	1280×1024	24×26	12×26	1272×1014	XGA
03/73	80×34	640× 480	16×14	8×14	640× 476	VGA
03/73	80×40	640× 480	16×12	8×12	640× 480	VGA
70/71	80×34	640× 480	16×14	8×14	640× 476	VGA
70/71	80×40	640× 480	16×12	8×12	640× 480	VGA
03/73	80×33	800× 600	16×18	8×18	640× 594	SVGA
03/73	80×42	800× 600	16×14	8×14	640× 588	SVGA
03/73	80×50	800× 600	16×12	8×12	640× 600	SVGA
70/71	100×33	800× 600	16×18	8×18	800× 594	SVGA

70/71	100×42	800× 600	16×14	8×14	800× 588	SVGA
70/71	100×50	800× 600	16×12	8×12	800× 600	SVGA
03/73	80×25	1040× 725	26×29	13×29	1040× 725	DA2
03/73	80×38	1040× 768	26×20	13×20	1040× 760	DA2
03/73	80×42	1024× 768	18×18	9×18	720× 756	DA2
70/71	128×42	1024× 768	16×18	8×18	1024× 756	DA2
03/73	80×25	1024× 768	24×30	12×30	960× 750	ET4K
03/73	80×38	1024× 768	24×20	12×20	960× 760	ET4K
03/73	80×33	800× 600	16×18	8×18	640× 594	ET4K
70/71	128×42	1024× 768	16×18	8×18	1024× 756	ET4K
70/71	100×33	800× 600	16×18	8×18	800× 594	ET4K

ドライバ欄に記載されている解像度は実際に表示する範囲です．
XGA：DPSXXGA.EXE（ただし，ビデオボードによりサポート範囲が異なる）
SVGA：DPSXSVGA.EXE／VGA：DPSXVGA.EXE／DA2：DPSXDA2.EXE／ET4K：DSPXET4K.EXE

◎英語モード時のビデオモード

ビデオモード	モード	色数	表示文字	解像度	フォント	ページ数
00/01	T	16	40×25	360× 400	9×16	8
02/03	T	16	80×25	720× 400	9×16	4
04/05	G	4	40×25	320× 200	8× 8	1
06	G	2	80×25	640× 200	8× 8	1
07	T	mono	80×25	720× 400	9×16	
0D	G	16	40×25	320× 200	8× 8	8
0E	G	16	80×25	640× 200	8× 8	4
0F	G	mono	80×25	640× 350	8×14	2
10	G	16	80×25	640× 350	8×14	2
11	G	2	80×30	640× 480		
12	G	16	80×30	640× 480		
13	G	256	40×25	300× 200		
以下，拡張						グラフィックアダプタ
18	T	16	132×44		8× 8	ET4000
19	T	16	132×25		8×14	ET4000
1A	T	16	132×28		8×13	ET4000
21	T	16	132×44	1188× 396	9× 9	SS24X
22	T	16	132×44		8× 8	ET4000
23	T	16	132×25		8×14	ET4000
24	T	16	132×28		8×13	ET4000
26	T	16	80×60		8× 8	ET4000
29	G	16	100×37	800× 600	8×16	ET4000
2A	T	16	100×40		8×15	ET4000
2E	G	256	80×30	640× 480	8×16	ET4000
30	G	256	100×37	800× 600	8×16	ET4000
37	G	16	128×48	1024× 768	8×16	ET4000
38	G	256	128×48	1024× 768	8×16	ET4000
41	T	16	80×34	720× 476	9×14	SS24X
47	T	16	132×28	1188× 448	9×16	SS24X
54	T	16	132×43	1188× 387	9× 9	ST ST24 SS24X
55	T	16	132×25	1188× 400	9×16	ST ST24 SS24X
58	G	16	100×75	800× 600	8×16	SS24X
5C	G	256	100×75	800× 600	8×16	SS24X
5D	G	16	128×48	1024× 768	8×16	SS24X
5E	G	256	80×25	640× 400	8×16	SS24X
5F	G	256	80×30	640× 480	8×16	SS24X
60	G	256	128×48	1024× 768	8×16	SS24X
62	G	32K		640× 480		SS24X
62	G	32K		800× 600		SS24X
64	G	16	160×64	1280×1024	8×16	SS24X
66	T	16	80×50	640× 400	8× 8	SS24X
67	T	16	80×43	640× 344	8× 8	SS24X
69	T	16	132×50	1056× 400	8× 8	SS24X
6A	G	16	100×75	800× 600	8×16	ST SS24X
6C	G	16	160×60	1280× 960	8×16	SS24X
72	G	16M		640× 480		SS24X

モード欄は，Tはテキストモード，Gはグラフィックモードです．
拡張部分は，各ビデオボードにより異なります．グラフィックボードの種類は，以下のビデオボードでの値です．
ST:Diamond STEALTH VRAM Hi-Color／ST24:Diamond STEALTH 24／SS24X:Diamond Speed Star 24X／ET4000:DFI VG-5000

ビデオBIOSを使用した画面モードの取得と設定

これらのビデオモードの取得・設定には，以下のビデオBIOS呼び出しが使用されます．また，これらの解像度をどう確認するかというと，下記のBIOSワークエリアを見るのがいちばんです．

アドレス	意味	サイズ
0040:0049	現在のビデオモード番号	1バイト
0040:004A	テキストの桁数	1ワード
0040:0084	テキストの行数ー1	1バイト
0040:0085	フォントの高さ（ドット数）	1ワード

ビデオBIOS呼び出しの機能番号（AH=0Fh）は，現在のビデオ状況を取得するための呼び出しで，現在のビデオモード番号，表示桁数，および活動ページ番号を返します．それぞれの情報は，BIOSワークエリアからもってきています．また，DOS/Vの標準ドライバでは，単一活動ページしかサポートしていないため，BHにはつねに0が設定されます．機能番号（AH=00h）は，ビデオモードを設定します．リスト2.1に，ビデオモードの設定および状況の取得関数をのせておきます．

ビデオモード取得関数では，ビデオBIOSワークエリアの値を見て，画面の横幅の文字数，画面の行数，フォントの高さを取得しています．40h:4Ah（便宜上，0:44Ahにしています）に画面の横幅の文字数が，40h:84h（同様に，0:484h）に画面の行数が，40h:85h（同様に，0:485h）にフォントの高さが入っています．ただし，画面の行数に関しては，

リスト2.1　ビデオモード取得・設定関数GETVMODE.ASM

```
include         std.inc

bios            segment at 0
                org     0449h
CurrentVmode    byte    ?                       ;現在のビデオモード
TextColumn      word    ?                       ;テキスト桁数
                org     0484h
TextLines       byte    ?                       ;テキスト行数-1
FontHeight      word    ?                       ;フォントの高さ


                .data
_ScreenColumn   word    ?                       ;画面の幅
_ScreenHeight   word    ?                       ;画面の高さ

                .code

;*****************************************************************
;*                                                               *
;*          ビデオモードの取得                                   *
;*          void        GetVideoInfo(VIDEO_INF *gvi);            *
;*                      VIDEO_INF *gvi  ビデオ情報構造体         *
;*          戻り値：      なし                                   *
;*                                                               *
;*  C言語側の定義                                                *
;*                                                               *
;*    typedef    struct {                                        *
;*                 uchar    CurrentVmode ;      //現在のビデオモード *
;*                 int      Column ;            //桁数           *
;*                 char     Lines ;             //行数           *
;*                 int      FontHeight ;        //フォントの高さ *
;*    }VIDEO_INF ;                                               *
;*                                                               *
;*    static    VIDEO_INF VideoInf ;            //初期ビデオモード格納 *
;*                                                               *
;*****************************************************************
GetVideoInfo    proc    uses es si di, gvi:PSTR
                if @model gt 3                  ;メモリモデルにより
                les     di, gvi                 ;構造体ｱﾄﾞﾚｽをセット
                else
                mov     di, gvi                 ;構造体ｱﾄﾞﾚｽをセット
                MOVSEG  es, ds
                endif
                push    ds
                sub     ax, ax
                mov     ds, ax
                assume  ds:bios
                mov     si, offset CurrentVmode ;現在のビデオモードを得る
                lodsb                           ;ビデオモードを取
                and     al, 7Fh                 ;最上位ビットをオフにする
                stosb                           ;ビデオモードを格納
                lodsw
                stosw                           ; 行の文字数
                push    ax
                mov     si, offset TextLines    ;画面の行数-1
                lodsb                           ;画面行数
                push    ax
                VIDEO   1D02h                   ;予約行数の取得
                pop     ax
                add     ax, bx                  ;予約行数を足す
                inc     ax
                stosb                           ;本当の画面行数
                movsw                           ;フォントの高さ
                pop     bx
                pop     ds
                assume  ds:@data
                mov     _ScreenColumn, bx       ;内部変数への保存
                mov     _ScreenHeight, ax       ;内部変数への保存
                ret
GetVideoInfo    endp

;*****************************************************************
;*                                                               *
;*          ビデオモードの設定                                   *
;*          void        SetVideoMode(int VideoMode);             *
;*          パラメータ:VideoMode    設定したいビデオモード       *
;*                                                               *
;*****************************************************************
SetVideoMode    proc    VideoMode:byte
                mov     al, VideoMode
                VIDEO   0
                ret
SetVideoMode    endp

                end
```

リスト2.2　実際のビデオモード取得関数GETVMOD2.ASM

```
include         std.inc

                .data
SetEv           db      'SETEV',0               ;常駐確認のためのID
SetEvApi        dd      ?                       ; SETEV APIエントリアドレス

                .code
;*****************************************************************************
;*                                                                           *
;*      実際のビデオモードの取得                                             *
;*      int         GetVideoMode2(void);                                     *
;*      出力:       ビデオモード                                             *
;*                                                                           *
;*****************************************************************************
GetVideoMode2   proc    uses bx cx dx es di
                mov     ax,0CACAh
                mov     di,offset SetEv
                int     2Fh                     ; SETEV常駐確認
                .if ax == 0 && bx == 'SE' && cx == 'TE' && dx == 5600h
                    mov     word ptr [SetEvApi+2],es
                    mov     word ptr [SetEvApi],di
                    mov     ah, 01h
                    call    SetEvApi            ;実際のVIDEO MODEの取得
                .else
                    VIDEO   0fh                 ; Get Current Video Status
                .endif
                xor     ah, ah
                ret
GetVideoMode2   endp

                end
```

かな漢字変換などが最下行を予約している場合がありますので，その行数を足しこむ必要があります．

この予約行数をチェックしているのが，「キーボードシフト表示域の状況の読み取り（AX=1D02h，INT 10h）」です．この結果，BXに予約行数が返ってきますので，さきほどの画面の行数の値に足し込みます．ただし，40h:84hの値は実際の行数－1になっていますので，忘れないでください．

ビデオモード設定関数は，単純にビデオBIOSを呼び出して，ビデオモードを設定しているだけです．ただし，V-Text環境の場合，過去のアプリケーションを動かすために，実際のビデオモードではない値をBIOSワークに設定する場合があります．この場合，そのビデオモードに戻そうとすると，本来のビデオモードと異なる場合があります．したがって，ビデオモード変更後，BIOSワークより解像度を確認し，元の状態と異なる場合は，ビデオモードを再度変更する必要があります．この場合問題になるのが，本当のビデオモードがわからないことです．

筆者自身は，今回の付録ディスクに添付したSETEVというプログラムを常駐させ，このAPIを利用して，実際のビデオモードを取得するようにしています（リスト2.2．本題とは関係ありませんが，SETEVは日本語モードと英語モードをプログラム，ディレクトリごとに動的に切り替えてくれる優れモノです）．

では，ビデオモードとして何を使用したらよいのでしょうか．テキストモードに関しては，まず，エミュレートCGAテキストモードなのか，エミュレート拡張CGAテキストモードなのかを考えなければいけません．前者の場合は仮想VRAMへのアクセスが，後者の場合は罫線や下線が使用できます．これらの必要性を検討します．

特定なビデオモードにしたい場合には，そのビデオモードに切り替えます．ただし，V-Textが使用できるかどうかは，後述のV-Text APIを使用するのがよいと思われます．ただし，古いV-TextドライバにはV-Text APIをサポートしていないものもあります．特定のビデオモードでなくてもよい場合には，つぎに現在のビデオモードを取得して，その値でプログラムが実行できるかを判定します．もし実行できる場合にはそのままプログラムを実行させます．実行できない場合で，プログラムがV-Textに対応している場合には，必要に応じてビデオモード03h,70h,71h,73hに切り替えます．このとき表示可能な行数・桁数をチェックします．また，V-Text APIを使用してフォントの品位・密度を切り替える場合もあります．

プログラム終了後は，元のビデオモードに戻しておくのがよいでしょう．ただし，前述のように実際のビデオモードと異なるものが取得されている場合もあるため，画面モードを元の値に戻した後，画面の行数・桁数を判定するようにしてください．もし，行数・桁数が異なる場合には取得されたビデオモードが異なっているか，またはV-Text APIで品位・密度を切り替えることが必要です．

SVGAの場合，ビデオモードはボードにより異なっていたため，業界標準を設定するために団体が設立されています．それがVESA（Video Electronics Standards Association）

表2.3　VESAで規定しているビデオモード

ビデオモード	モード	色数	表示文字	解像度	グラフィックアダプタカード		
100	G	256		640× 400	SS24X		ST24
101	G	256		640× 480	SS24X	ST	ST24
102(6A)	G	16		800× 600	SS24X	ST	ST24
103	G	256		800× 600	SS24X	ST	ST24
104	G	16		1024× 768	SS24X	ST	ST24
105	G	256		1024× 768	SS24X		ST24
106	G	16		1280×1024	SS24X		ST24
107	G	256		1280×1024			
108	T	16	80×60				
109	T	16	132×25		SS24X		ST24
10A	T	16	132×43		SS24X		ST24
10B	T	16	132×50				
10C	T	16	132×60				
==VBE V1.2+							
10D	G	32K		320× 200			
10E	G	64K		320× 200			
10F	G	16K		320× 200			
110	G	32K		640× 480			
111	G	64K		640× 480			
112	G	16K		640× 480			
113	G	32K		800× 600			
114	G	64K		800× 600			ST24
115	G	16K		800× 600			
116	G	32K		1024× 768			
117	G	64K		1024× 768			
118	G	16K		1024× 768			
119	G	32K		1280×1024			
11A	G	64K		1280×1024			
11B	G	16K		1280×1024			
==S3 OEM							
201	G	256		640× 480		ST	ST24
202	G	16		800× 600		ST	ST24
203	G	256		800× 600		ST	ST24
204	G	16		1024× 768		ST	ST24
205	G	256		1024× 768		ST	ST24
206	G	16		1280× 960		ST	ST24
208	G	16		1280×1024		ST	ST24
211(111)	G	64K		640× 480			ST24
212(112)	G	16M		640× 480			ST24
301(110)	G	32K		640× 480		ST	ST24

また，VESAの機能呼び出し方法と，各機能は以下のとおりです．

【呼び出し方法】

入力：	AH=4Fh	SVGA拡張機能
	AL=00-08h	VESA機能番号
	INT　10h	
出力：	AL=4Fh	機能はサポートされている
	AL<>4Fh	機能は未サポート
	AH=00h	機能は正常に実行された
	AH=01h	機能は失敗した

機能番号	意　味
00h	SVGA情報の取得
01h	SVGAモード情報の取得
02h	SVGAビデオモードの設定
03h	SVGAビデオモードの取得
04h	SVGAビデオ状況の保管・復元
05h	SVGAビデオメモリウィンドウの制御
06h	論理スキャンラインレングスの取得・設定
07h	ディスプレイスタートの取得・設定
08h	DACパレットコントロールの取得・設定

で，米国のビデオボードメーカーとモニターメーカーが約120社集まっています．

とくにDOS/Vプログラミングに関連するものとしては，ビデオモードの固定化やローレベル機能呼び出しの内容・方法の統一化などが決定されています．

VESAで規定しているビデオモードには表2.3のものがあります．また，呼び出し方法なども表2.3に示しておきました．

2.3 エスケープシーケンス文字を使用した画面表示

DOS汎用のプログラムを作成する場合には，当然のことながら，機種依存しているビデオBIOSを使用することはできません．しかし，エスケープシーケンス文字を使用してきめ細かい画面制御を行うプログラムを作成することができます．ここでは，エスケープシーケンス文字に関して解説します．

◎AT互換機で，エスケープシーケンスを使用する場合には，ANSI.SYSまたは同等機能をもつドライバが必要です．

◎PC-9801と比較すると，サポートしているエスケープシーケンス文字の種類は少ないです．

エスケープシーケンスを使用する場合，前述したとおり，ANSI.SYSまたは同等の機能をもったドライバの組み込みが必要です．したがって，ANSI.SYSを使用する場合，それが組み込まれているかを確認し，組み込まれていない場合はANSI.SYSを組み込むようにメッセージを表示したほうが親切でしょう．ANSI.SYSの組み込みを確認するためには，リスト2.3の関数を使用します．この組み込み確認用の割り込みは，DOS 5.0以降に公開になっており，DOS　4.0以下では非公開機能でした．しかし，筆者の確認した範囲内ではDOS 4.0以下でも，とくに使用上は問題はありませんでした．

ANSI.SYSを組み込むことにより，以下のような画面の表示の制御とキーボードのキーの機能を制御することができます．

◎画面上の任意の位置へカーソルを移動する．

◎コマンドプロンプトの位置を変更する．

◎画面のモード，テキストの色，画面の背景色を変更する．

リスト2.3　ANSI.SYS組み込み確認関数ISANSI.ASM

```
include         std.inc

                .code

;*******************************************************************************
;*                                                                             *
;*        ANSI.SYS組み込み確認関数                                             *
;*        int        isAnsiSys(void);                                          *
;*        戻り値：     = 0 : 組み込まれていない                                *
;*                    != 0 : 組み込まれている                                  *
;*                                                                             *
;*******************************************************************************
isAnsiSys       proc
                xor     bx, bx                  ; AVATAR.SYSのIDと重ならないよう
                INT2F   1a00h                   ; ANSI.SYS組み込みﾁｪｯｸID
                                                ;   AL=0FFhで戻ってきた場合，
                                                ;   ANSI.SYSが組み込まれている
                .if al != 0FFh                  ; ANSI.SYS組み込まれていない場合
                    xor     ax, ax
                .endif
                ret
isAnsiSys       endp

                end
```

◎キーにコマンドを割り当てたり，特別な文字列を登録したりする．

エスケープシーケンス文字は，ESC文字（1Bh）と“[”からはじまり，これに続く文字はパラメータ（ない場合もあります．複数のパラメータが必要な場合には，次の例のように“;”で区切ります）で，最後に1文字のコマンド文字があります．このコマンド文字は大文字・小文字の区別があります．また，スペースはエスケープシーケンス文字群の終わりを示しますので途中に入れることはできません．

ESC[1;1H

また，DOS/Vで使用できるエスケープシーケンス文字に関しては表2.4を参照してください．通常のPC-9801や国産機種と比較すると，使用できるエスケープシーケンス文字が若干少ないので注意してください．

表2.4　DOS/Vエスケープシーケンスサポート表

◎カーソルの移動

ESC [Pl ; Pc H	カーソル位置の設定(Pl=行, Pc=桁)
ESC [Pl ; Pc f	同上 パラメータが指定されていない場合には，1行目1桁目に移動します．
ESC [Pn A	カーソルを上に移動(Pn=行数)
ESC [Pn B	カーソルを下に移動(Pn=行数)
ESC [Pn C	カーソルを右に移動(Pn=桁数)
ESC [Pn D	カーソルを左に移動(Pn=桁数) パラメータが指定されていない場合には，1行または1桁だけ移動します．また，画面の該当方向の端にカーソルがある場合には，このコマンドは無視されます．
ESC [s	現在のカーソル位置の保存
ESC [u	ESC[sコマンドで保管されたカーソル位置を復元

◎消去関連

ESC [2 J	画面全体を消去し，カーソルを1行目1桁目に移動
ESC [K	カーソル位置の文字を含めて，行右端まで消去

◎画面属性の変更

ESC [Ps ; ...;Ps m　グラフィック属性の設定

		テキストモード		グラフィックモード	
Ps	テキスト属性	カラー	モノクロ *1	カラー	モノクロ
0	画面属性解除	○	○	○	○
1	高輝度	○	○	○	×
4	下線	×	○	×	×
5	ブリンク *2	○	○	×	×
7	リバース(絶対)	○	○	×	×
8	シークレット	○	○	○	×
30	黒色の前景色	○	×	○	×
31	赤色の前景色	○	×	○	×
32	緑色の前景色	○	×	○	×
33	黄色の前景色	○	×	○	×
34	青色の前景色	○	×	○	×
35	紫色の前景色	○	×	○	×
36	水色の前景色	○	×	○	×
37	白色の前景色	○	×	○	×
40	黒色の背景色	○	×	○	×
41	赤色の背景色	○	×	○	×
42	緑色の背景色	○	×	○	×
43	黄色の背景色	○	×	○	×
44	青色の背景色	○	×	○	×
45	紫色の背景色	○	×	○	×
46	水色の背景色	○	×	○	×
47	白色の背景色	○	×	○	×

*1：日本語モードにはモノクロのテキストモードはありません．
*2：英語モードでのみサポートされます．

表2.4（続き）

◎ビデオモードの変更			
ESC [= Ps h	ビデオモードの設定		
	0	CGA 40×25	モノクロテキスト
	1	CGA 40×25	カラーテキスト
	2	CGA 80×25	モノクロテキスト
	3	CGA 80×25	カラーテキスト
	4	320×200	4色カラーグラフィック
	5	320×200	モノクログラフィック
	6	640×200	モノクログラフィック
	7	右マージンラップモード	
	13	320×200	カラーグラフィック
	14	640×200	16色カラーグラフィック
	15	640×350	モノクログラフィック
	16	640×350	16色カラーグラフィック
	17	640×480	モノクログラフィック
	18	640×480	16色カラーグラフィック
	19	320×200	256色カラーグラフィック
	112	V-Text CGA	カラーテキスト
	113	V-Text 拡張CGA	カラーテキスト
	114	640×480	16色カラーグラフィック
	115	拡張CGA 80×25	カラーテキスト
ESC [= Ps l	モードのリセット		
	7	右マージン非ラップモード	
	7以外	上記と同じ	
◎キーボードの割り当て変更			
ESC [Pc ; Ps ; ... ; Pc ; Ps p	キーボード再割当(Pc=コード, Ps=文字列)		

※エスケープシーケンス文字列の途中にあるブランクは，あくまでも見やすくするために入れているだけで，実際には入れないでください．また，ESCの表記は1Bhを意味しています．PlやPcなどは，実際には10進数でパラメータを指定します．最後の1文字はそのままの文字を使用してください．

テキストモード画面で，エスケープシーケンス文字を使用して画面属性の変更を行っている場合にスクロールが発生すると，スクロールしたあとの行の属性は，スクロール前の最下行の1桁目の画面属性がそのまま使用されます（これは，エスケープシーケンスとは直接関係のないスクロール機能の仕様なのですが）．したがって，その行だけのつもりで画面属性を変更し，文字出力を行い，元の画面属性に戻したつもりでも，実際は次の行にまで画面属性が反映してしまう場合があります．こういった場合には，その行の先頭文字（ブランクにするとよいと思います）だけは普通輝度で表示し，2文字目から画面属性を変更するようにするとよいでしょう．

キーボードの割り当て変更は，DOSの文字入力で取得される文字コードを変更するものです．たとえば，

ESC[0;59;"DIR"p

では，[F1]キー（0;59がそれを意味しています）を押すと，DIRが入力されたのと同じ意味をもちます．しかし，この機能は現在の値がわからないため，よほどのケースでないと使用されていません．

機能の詳細に関しては，『IBM DOSバージョンJ5.0ユーザーズ・ガイド』のANSI.SYSデバイスドライバの解説部分に記載されています．

2.4 カーソル制御

カーソルに関連するものとしては，カーソルの形状の取得・設定（表示・非表示を含む）とカーソル位置の取得・設定があります．これらに関するビデオBIOSについて解説します．

◎カーソルの形状および位置に関しては，BIOSワークエリアにありますが，通常はビデオBIOSを使用して，取得・設定を行います．

◎グラフィック画面ではカーソルは表示されません．

◎標準では，カーソルの明滅や全角カーソルをサポートしていません．アプリケーションソフトやV-Textドライバによってはサポートしているものもあります．

まず，カーソルまわりから説明していきます．カーソル制御は基本的にソフトウェアで行っており（一部，V-Textドライバではハードウェアで行っているものもあります），画面表示を遅くしている原因となっています．実際には，グラフィック画面をカーソル形状でXORしています．この場合，カーソルはフルサイズにしないで，もっとも小さくすると見違えるように速くなる場合がありますので一度試してください．また，まとめて文字の表示を行う際もカーソルを消してから書き込むようにすると，表示速度が向上する場合があります．

以下にカーソルの制御について説明しますが，その前に画面上での位置関係と形状について図2.4にまとめておきます．

カーソル関連のビデオBIOSには以下のものがあります．

カーソルの位置

桁位置進行方向→

0 n

0

行位置進行方向↓

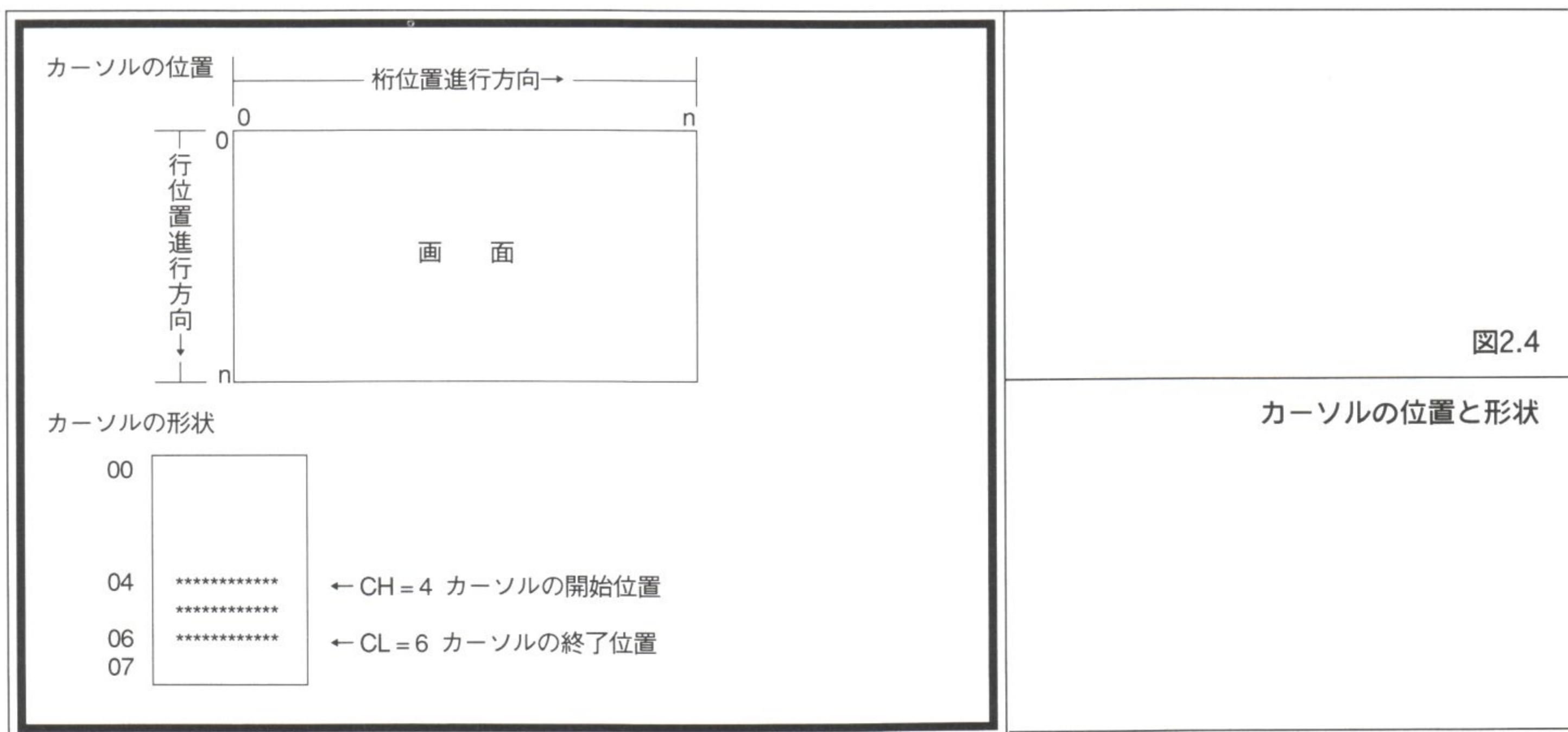
画　面

n

カーソルの形状

```
00  |              |
04  | ************ |  ←CH＝4 カーソルの開始位置
    | ************ |
06  | ************ |  ←CL＝6 カーソルの終了位置
07  |              |
```

図2.4

カーソルの位置と形状

カーソル形状の設定

カーソル形状を設定する場合には，以下のビデオBIOSを使用します．

```
MOV     CX,カーソルの形状
VIDEO   01h
```

本来，カーソルの大きさは，フォントの高さと同じ範囲内で指定します．しかし，ビデオBIOSでのカーソル形状は，画面のフォントサイズにかかわらず，CGA互換になるように，0～7の値で，文字ボックス内のカーソルの開始位置，終了位置を指定します．したがって，同じ値を指定してもビデオドライバによって多少位置が変わる場合があります（気にすることはありません）．ただし，CHのビット6-5が01の場合（通常は，CX=2000h）には，カーソルは表示されません．また，カーソルはテキスト画面でしか表示されませんので注意してください．

カーソル位置の設定

カーソル位置は以下のように設定します．

```
MOV     DH, 行番号    0からの相対指定
MOV     DL, 桁番号    0からの相対指定
XOR     BH, BH        活動ページ番号
VIDEO   02h
```

カーソル位置指定は画面の左上が（0,0）となります．ただし，リスト2.4のカーソル関連の関数群では内部的に1を引くようにしていますので，画面の左上が（1,1）となっています．

カーソル形状・位置の取得

現在のカーソルの形状，位置を読み取るには，以下のようにします．

```
XOR     BH, BH        DHに現在の行位置（0からの相対）
VIDEO   03h
```

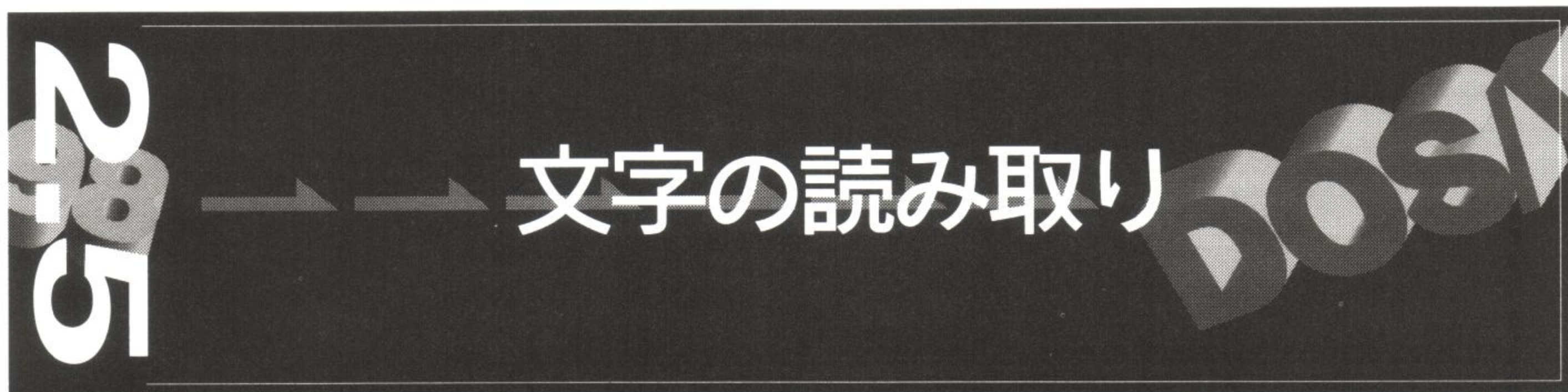
リスト2.4　カーソル形状・位置の設定CURSOR.ASM

```
include         std.inc

                .code

;*************************************************************************
;*                                                                       *
;*          カーソル形状の設定                                           *
;*          void        SetCursorType(int start, int endpos);            *
;*          パラメータ:start  = 開始位置 (0-7) 20hのとき, 非表示         *
;*                      endpos = 終了位置 (0-7)                          *
;*                                                                       *
;*************************************************************************
SetCursorType   proc    start:byte, endpos:byte
                mov     ch, [start]                 ;開始位置
                mov     cl, [endpos]                ;終了位置
                VIDEO   01h
                ret
SetCursorType   endp

;*************************************************************************
;*                                                                       *
;*          カーソル位置の設定                                           *
;*          void        SetCursorPos(int row, int col);                  *
;*          パラメータ:row   = 行番号 (1-)                               *
;*                      col   = 桁番号 (1-)                              *
;*                                                                       *
;*************************************************************************
SetCursorPos    proc    row:byte, col:byte
                mov     dh, [row]                   ;行番号
                mov     dl, [col]                   ;桁番号
                sub     dx, 101h                    ;行桁の補正
                xor     bh, bh                      ;活動ページ番号
                VIDEO   02h
SetCursorPos    endp

;*************************************************************************
;*                                                                       *
;*          カーソル位置・形状の保管                                     *
;*          void        SaveCursor(void)                                 *
;*                                                                       *
;*          カーソル位置・形状の復元                                     *
;*          void        RestoreCursor(void)                              *
;*                                                                       *
;*************************************************************************
SaveCur         dw      ?                           ;カーソル形状保存用内部変数
SaveCurPos      dw      ?                           ;カーソル位置保存用内部変数

SaveCursor      proc
                xor     bh, bh                      ;活動ページ番号
                VIDEO   03h                         ;カーソル位置・形状の読み取り
                mov     cs:SaveCur, cx
                mov     cs:SaveCurPos, dx
                ret
SaveCursor      endp

RestoreCursor   proc
                mov     cx, cs:SaveCur
                VIDEO   01h                         ;カーソル形状の設定
                mov     dx, cs:SaveCurPos
                xor     bh, bh                      ;活動ページ番号
                VIDEO   2                           ;カーソル位置の設定
                ret
RestoreCursor   endp

                end
```

DXに現在のカーソル位置が，CXに現在のカーソル形状が戻ってきます．

これらのビデオBIOS機能をCから呼び出す関数群をリスト2.4にのせておきます．

リスト2.4の関数群を使用することにより，カーソルの保管・復元および形状の設定，位置の設定が可能となります．とくにエラーチェックは一切行っていません．また，カーソル位置を取得したい場合には，SaveCursorを修正して，CXの値を親に戻すようにすれば可能です．

2.5 文字の読み取り

画面上の文字・属性を読み取るためには，ビデオBIOSを使用する方法と仮想VRAMを直接アクセスする方法の2種類に大別できます．ここでは，ビデオBIOSを使用する方法に関して解説します．

◎ビデオBIOSには，「現在のカーソル位置の文字と属性の読み取り」と「文字ブロックの読み取り」の2種類があります．

◎前者は，ビデオモードに関連せずにビデオBIOS経由で書いた文字・属性が読み取れます．ただし，エミュレート拡張CGAテキストモードの場合は，取得できる属性は色属性だけで，拡張属性（罫線や下線）は取得できません．

画面上の文字・属性を読み取るためには，以下のBIOS呼び出しを使用します．

カーソル位置の文字と属性の読み取り（AH=08h）

この機能では，現在のカーソル位置上の文字・属性しか読み取ることができません．したがって，複数の文字を読み取るためには，1文字ずつカーソルの位置づけ（機能コードAH=02h）を行ってから，

```
BH = 0         活動ページ番号
VIDEO   08h    カーソル位置の文字と属性の読み取り
```

として，文字の読み取りを行わなければなりません．戻り値としては，AHに属性が，ALに文字コードが戻ります．活動ページ番号には，必ず0をセットします．また，この機能では，エミュレート拡張CGAテキストモードのときに取得できるのは文字と色属性（前景色，背景色）だけで，拡張属性（罫線，下線）は取得できません．

この機能を実行するときに，カーソルを表示状態にしておくと，実際にカーソルが移動して画面が見にくくなりますので，カーソル位置・形状の保管を行い，カーソルを消去してから実行するようにしてください．

この機能を応用した表示画面上の文字ブロックの読み取り関数をリスト2.5に示します．

この機能はIBM PC系では共通のため，速度さえ気にしなければ，どの機種でも動作します．また，グラフィック画面でも，本当にグラフィックとして書いた文字は別ですが，

リスト2.5　　**カーソル位置の文字と属性の読み取りを使った文字ブロックの読み取り**

```
include         std.inc

                .code

;****************************************************************************
;*                                                                          *
;*          文字列の読み取り                                                *
;*          void        GetString(int row, int col, int leng, char *strptr)  *
;*          パラメータ:row    = 行番号 (1-)                                 *
;*                     col    = 桁番号 (1-)                                 *
;*                     leng   = 長さ                                        *
;*                     strptr   = バッファへのポインタ                      *
;*                                                                          *
;****************************************************************************
bios            segment at 0
                org     044Ah
TextColumn      word    ?                       ; テキスト桁数

GetString       proc    uses bx cx dx es di,¥
                                row:byte, col:byte, leng:word, strptr:PBYTE
                xor     bh,bh
                VIDEO   03h                     ; 現在のカーソル位置状態の保存
                PUSHM   <cx,dx>                 ; カーソル状態の保存
                mov     cx,2000h                ; カーソルを非表示にする
                VIDEO   01h                     ; カーソル形状の設定
                mov     dh, [row]               ; 行番号
                mov     dl, [col]               ; 桁番号
                sub     dx, 101h                ; 行桁の補正
                mov     cx, [leng]              ; 長さ
                if @model gt 3                  ; large modelなど
                les     di, [strptr]            ; 保管文字列の先頭アドレス
                else                            ; small model など
                mov     di, [strptr]            ; 保管文字列の先頭アドレス
                MOVSEG  es, ds
                endif
                push    ds
                xor     ax, ax                  ; ds をBIOSワークエリアに
                mov     ds, ax                  ;    セットする
                assume  ds:bios
                mov     bl, byte ptr TextColumn ; 画面の桁数
                pop     ds
                .repeat
                    xor     bh, bh              ; 活動ページ番号
                    VIDEO   02h                 ; カーソル位置の設定
                    VIDEO   08h                 ; 文字と属性の取得
                    stosw                       ; AH=属性, AL=文字 を保存
                    inc     dl                  ; 1 桁進める
                    .if dl == bl                ; 画面の右端のとき
                       inc     dh               ; 次の行へ
                       xor     dl, dl           ; 先頭桁へ
                    .endif
                .untilcxz
                POPM    <dx,cx>                 ; カーソル状態を復帰させる
                VIDEO   02h                     ; カーソル位置をdxに設定
                VIDEO   01h                     ; カーソル形状をcxに設定
                ret
GetString       endp

                end
```

ビデオBIOS経由で書いた文字，属性は読み取ることができます．もし，DOS/Vだけを対象とする場合は，次の文字ブロックの読み取りを使用するほうがパフォーマンスがよいものとなります．ただし，文字ブロックの読み取り機能はDOS/V独自の拡張であり，ほかのIBM PC系DOSとは互換性がなくなります．

文字ブロックの読み取り（AH=13h）----------------------

表示画面に対する文字ブロック単位の読み書きは，機能コード（AH=13h）としてまとめられています．本来のAT系BIOSでは，文字ブロックの書き込みだけなのですが，DOS/Vでは拡張属性に対応するために文字ブロックの読み取り・書き込み機能を追加しています．このうち，文字ブロックの読み取りに関しては，サブ機能コード（AL=1xh）によって実行されます．AL=10hが，画面モード03hおよび70hのエミュレートCGAテキストモードに対応し，AL=11hが画面モード73hおよび71hのエミュレート拡張CGAテキストモードに対応しています．

表示画面から文字ブロック単位で読み取るためには，以下のようにします．

```
BH = 0          活動ページ番号，つねに0
DH =            行位置，画面の一番上端が0
DL =            桁位置，画面の一番左端が0
CX =            文字数
ES:BP           読み取りバッファの先頭アドレス
AL =            サブ機能コード（表2.5参照）
VIDEO   13h     文字ブロックの読み取り
```

表2.5　　**文字列ブロックの読み取り**
《ALで指定されるサブ機能コード》

AL	機　能
10h	以下の形式でES:BPで指定したバッファへ文字列を読み取る 文字 属性 文字 属性 文字 属性 文字 属性
11h	以下の形式でES:BPで指定したバッファへ文字列を読み取る 文字 属1 属2 属3 文字 属1 属2 属3

ALには，画面モードに応じて，10h,11hを指定すると，表2.5の形式で指定したバッファアドレスに読み取られます．

この機能は現在のカーソル位置とは無関係で，実行後もカーソル位置は移動しません．サブ機能コード（AL=10h）を使用した文字ブロック読み取り関数をリスト2.6に示します．

リスト2.6　文字ブロックの読み取り機能を使用した文字ブロックの読み取りGETSTR.ASM

```
include         std.inc

                .code

;*******************************************************************
;*                                                                 *
;*        文字列の読み取り                                         *
;*        void       GetString(int row, int col, int leng, char *strptr)  *
;*        パラメータ:row    = 行番号 (1-)                          *
;*                   col    = 桁番号 (1-)                          *
;*                   leng   = 長さ                                 *
;*                   strptr = バッファへのポインタ                 *
;*                                                                 *
;*******************************************************************
GetString       proc    uses bx cx dx es,¥
                                row:byte, col:byte, leng:word, strptr:PBYTE
                assume  ds:@data
                mov     dh, [row]               ; 行番号
                mov     dl, [col]               ; 桁番号
                sub     dx, 101h                ; 行桁の補正
                mov     cx, [leng]              ; 長さ
                if @model gt 3                  ; large model など
                les     bp, [strptr]            ; 保管文字列の先頭アドレス
                else                            ; small model など
                mov     bp, [strptr]            ; 保管文字列の先頭アドレス
                MOVSEG  es, ds
                endif
                xor     bh, bh                  ; 活動ページ番号
                VIDEO   1310h                   ; 文字列の読み取り
                ret
GetString       endp

                end
```

2.6 文字の書き込み

画面上に文字・属性を表示するには，読み取りと同様に，ビデオBIOSを使用する方法と仮想VRAMを直接アクセスする方法の2種類に大別できます．ここでは，ビデオBIOSを使用する方法に関して解説します

◎ビデオBIOSには，たれ流し的に出力する「テレタイプ方式の書き込み」，カーソル位置に出力する「現在のカーソル位置へ文字を書き込む」，「現在のカーソル位置へ文字と属性を書き込む」と「文字列の書き込み」の4種類があります．

◎前の2つのBIOS呼び出しは文字だけを，後ろの2つのBIOS呼び出しは文字と属性を出力します．

◎文字列の書き込みにも，4つのタイプがあります．

◎全角文字を書き込む場合，文字列の書き込みでは1回のBIOS呼び出しで，それ以外の場合は必ず1文字目と2文字目を連続して書き込みます．

ビデオBIOSによる文字の出力には大きく3種類，実際には4種類の方法があります．では，これらの違いは，どこにあるのでしょうか．

テレタイプ式文字の書き込み（AH=0Eh）

この機能は現在のカーソル位置に1文字を書き込み，カーソルを1つ進めます．書き込んだ位置が最後の行の，最後の桁である場合，BIOSは画面全体を1行スクロールさせます．このとき，グラフィックモードの場合，最下行は黒のスペースで埋められますが，テキストモードの場合は，スクロールする前の最下行の1文字目の属性を使用して，スペースで埋

リスト2.7　テレタイプ式文字の書き込みPUTSTR.ASM

```
include         std.inc

                .code

;*****************************************************************************
;*                                                                           *
;*        テレタイプ式文字の書き込み                                          *
;*        void        PutString(char *strptr)                                *
;*        パラメータ:strptr    = バッファへのポインタ                          *
;*                                                                           *
;*****************************************************************************
PutString       proc    uses ds si,strptr:PBYTE
                if @model gt 3                      ; large model など
                lds     si, [strptr]                ; 文字列の先頭アドレス
                else                                ; small model など
                mov     si, [strptr]                ; 文字列の先頭アドレス
                endif
                .while 1
                    lodsb                           ; 次の1文字
                    .break .if al == 0              ; 文字列の終わり
                    VIDEO   0Eh                     ; 1 文字出力
                .endw
                ret
PutString       endp

                end
```

められます．また，復帰（0Ch），改行（0Ah），バックスペース（08h），ベルコード（07h）は制御コードとして処理され，画面への表示は行われません．

テレタイプ方式の書き込みを行う場合には，以下のようにします（リスト2.7）

BH = 0	活動ページ番号，つねに0
AL =	文字コード
BL =	文字属性（グラフィック時のみ）
VIDEO　0Eh	テレタイプ式文字の書き込み

現在のカーソル位置へ文字を書き込む（AH=0Ah）／文字と属性を書き込む（AH=09h）--

つぎに，AH=09h,AH=0Ahですが，これらは属性を処理するかどうかに違いがあるだけです．最大の特徴は，現在のカーソル位置にしか出力できず，かつ，カーソルは移動しないため，前述のカーソル位置の文字読み取りと同様に，カーソル位置の設定，文字出力のくり返しになります．したがって，画面出力のパフォーマンスは悪いものになります．ただし，1行消去時（AL=20h,CX=80など）などに使えば，1回のBIOS呼び出しで処理できますし，属性も設定（AH=09hの場合）できます．

BH = 0	活動ページ番号，つねに0
AL =	文字コード
CX =	書き込む回数
VIDEO　0Ah	現在のカーソル位置へ文字を書き込む

BH = 0	活動ページ番号，つねに0
AL =	文字コード
CX =	書き込む回数
BL =	文字属性
VIDEO　09h	現在のカーソル位置へ文字と属性を書き込む

文字列の書き込み（AH=13h）------------------------------

この機能はALにサブ機能コードを指定することによって，さまざまな方法で文字列の書き込みを行うものです（リスト2.8）．

BH = 0	活動ページ番号，つねに0
DH =	行位置，画面の一番上端が0
DL =	桁位置，画面の一番左端が0
BL =	文字属性，サブ機能コードが00h,01hの場合に必要

CX =　　　　文字数
ES:BP　　　　書き込みバッファの先頭アドレス
AL =　　　　サブ機能コード（表2.6参照）
VIDEO　13h　　文字列の書き込み

お薦めは，AH=13hでしょう．とくにカーソル位置の移動を考えない場合は，AL=20hなどを使うと，パフォーマンスもよく，特殊文字（改行，復帰，BS，ベル）も処理されず，画面範囲を越した場合には単純に破棄され，スクロールもおきません．逆にこういったものを処理したい場合には，AL=0xh系を使用するのがいいと思います．

表2.6　文字列の書き込み
《ALで指定されるサブ機能コード》

AL	機　能	カーソル
00h	BLで指定された属性で，ES:BPの文字列を書き込む．	移動しない
01h	文字 文字 文字 文字	移動する
02h	以下の形式でES:BPに格納された文字列を書き込む．	移動する
03h	文字 属性 文字 属性 文字 属性 文字 属性	移動しない
20h	以下の形式でES:BPに格納された文字列を書き込む． 文字 属性 文字 属性 文字 属性 文字 属性	移動しない
21h	以下の形式でES:BPに格納された文字列を書き込む． 文字 属1 属2 属3 文字 属1 属2 属3	移動しない

リスト2.8　文字列の表示TEXTCTRL.ASM

```
include         std.inc

                .code

;****************************************************************************
;*                                                                          *
;*          文字列の表示                                                    *
;*          void      PutStringAbs(x,y,dstr,atr);                           *
;*          パラメータ： x ,y  表示開始座標                                 *
;*                       dstr  文字列                                       *
;*                       atr   画面属性                                     *
;*          戻り値：     なし                                               *
;*                                                                          *
;****************************************************************************
PutStringAbs    proc    uses ds es si di, x:byte, y:byte, dstr:PBYTE, atr:byte
                assume  ds:@data
                if @model gt 3
                lds     si,[dstr]               ;文字列の先頭アドレス
                else
                mov     si,[dstr]               ;文字列の先頭アドレス
                endif
                xor     cx,cx
                .while byte ptr ds:[si] != 0    ;文字列の長さを求める
                    inc     cx
                    inc     si
                .endw
                .if cx != 0                     ;文字列長がある場合
                    mov     bl,[atr]            ;表示属性
                    xor     bh,bh               ;活動ページ
                    mov     dl,[x]              ;桁
                    mov     dh,[y]              ;行
                    sub     dx,101h             ;行，桁の補正
                    if @model gt 3
                    les     bp,[dstr]           ;文字列のアドレス
                    else
                    MOVSEG  es,ds
                    mov     bp,[dstr]           ;文字列のアドレス
                    endif
                    VIDEO   1300h               ;文字列表示(ｶｰｿﾙ移動なし)
                .endif
                ret
PutStringAbs    endp

                end
```

2-7　仮想VRAMへの直接表示

前にも触れましたが，エミュレートGCAテキストモード（ビデオモード03hと70h）には，英語モードで直接テキストVRAMをアクセスしているプログラムを移植しやすくするために，仮想VRAMが存在しています．仮想VRAMをアクセスするために，TopViewイン

ターフェースと呼ばれる仮想VRAMインターフェースを用意しています．この仮想VRAM構造はどうなっているのでしょうか．VRAM構成は，PC-9801と違って，文字・属性・文字・属性……のように，1バイトごとに交互にならんでいます．コード体系も，ASCII＋シフトJISコードといった通常のコードをそのまま書きこめるため，コード変換を行う必要はありません．

このような構造になっているため，速度が要求されない表示にはビデオBIOSを使用し，速度が必要な場合は仮想VRAMに直接書き込み，あとで仮想VRAMからの描画BIOS機能を呼び出すようにするのがよいと思われます．また，BIOSを使用する場合にも，なるべくまとめて呼び出し，BIOSを使用する回数を少なくしてください．

この仮想VRAMを使用するためには，まずは，仮想VRAMの実際のアドレスを取得しなければいけません．このためには，以下のビデオBIOS機能を使用します．

ES:DI = B800h:0000h 実テキストVRAMの先頭アドレス
VIDEO FEh 仮想VRAMアドレスの取得

戻ってきたES:DIが仮想VRAMの実アドレスです．DOS/V日本語モードのビデオモード03hと70h以外の場合には，このBIOS呼び出しは無視され，ES:DIには元の値がそのままセットされています．また，COMPAQ DOS/Vでは，80386のページング機能を使用して，仮想VRAMをB000h:0000hに割り当てているため，この場合もそのままの値が返ってきます．ただし，COMPAQ DOS/Vの場合には，後述の再表示機能を使用しなくても表示するように変更されています．機能的には便利なのですが，若干パフォーマンスが落ちてしまいます．

Cから使用する場合には，ビデオの初期化として，リスト2.9のSetVramSegment関数を必ず呼び出し，内部変数に仮想VRAMのセグメントアドレスを記憶させます．Cから仮想VRAMをfarポインタで操作できるように，仮想VRAMアドレスを戻すようにしています．

戻ってきたES:DIが仮想VRAMのアドレスです．注意しなければいけないのが，戻ってきたDIが0であるとは限らないことです．しかし，VRAMアドレス計算をする場合，64KB

リスト2.9　ビデオ初期化処理SETVSEG.ASM

```
include                 std.inc

                .data
_VideoSegment   dword    ?                      ;仮想VRAMセグメント

                .code

;*******************************************************************************
;*                                                                             *
;*          ビデオ初期化処理                                                   *
;*          void far *SetVramSegment(void);                                    *
;*          戻り値：     仮想VRAMのアドレス                                    *
;*                                                                             *
;*******************************************************************************
SetVramSegment  proc     uses es di
                assume   ds:@data
                mov      ax,0B800h                   ; es:di =実ビデオバッファの
                mov      es,ax                       ;         アドレス
                xor      di,di
                VIDEO    0FEh                        ;仮想VRAMｾｸﾞﾒﾝﾄの取得
                mov      word ptr _VideoSegment,di   ;内部変数に保管
                mov      word ptr _VideoSegment+2,es ;内部変数に保管
                mov      ax, di                      ; es:di仮想VRAMのアドレス
                mov      dx, es
                ret
SetVramSegment  endp

                end
```

リスト2.10　画面表示の更新REWRITE.ASM

```
include                 std.inc

                .code

;*******************************************************************************
;*                                                                             *
;*          画面表示の更新                                                     *
;*          void     RewriteVideo(void far *VramAdr, unsigend Leng);           *
;*          パラメータ： VramAdr  更新する仮想VRAMの先頭アドレス               *
;*                       Leng     書き込む文字数                               *
;*          戻り値：     なし                                                  *
;*                                                                             *
;*******************************************************************************
RewriteVideo    proc     uses cx es di, VramAdr:far ptr byte, Leng:word
                les      di, VramAdr                 ; 更新する仮想VRAMの先頭アドレス
                mov      cx, Leng                    ; 書き込む文字数
                VIDEO    0FFh                        ; 画面表示の更新
                ret
RewriteVideo    endp

                end
```

リスト2.11　　画面の再表示REWRITE.ASM

```
include                 std.inc

                .code

;*************************************************************************
;*                                                                       *
;*          画面の再表示                                                 *
;*          void        ScreenRewrite(y1,y2);                            *
;*          パラメータ： y1    開始行                                     *
;*                       y2    終了行                                     *
;*          戻り値：     なし                                             *
;*                                                                       *
;*************************************************************************
ScreenRewrite   proc    uses bx cx es di, y1:word, y2:word
                assume  ds:@data
                mov     ax, y1                  ;開始行
                sub     bx, bx                  ;桁
                call    _CalcVramAddr           ;仮想VRAMアドレスの計算
                                                ; ES:DI =更新したいビデオバッ
                                                ;        ファー内の先頭アドレス
                mov     cx, y2                  ;終了行
                sub     cx, ax                  ;再表示する行数-1
                mov     ax, cx                  ;
                inc     ax                      ;再描画する行数
                mul     _ScreenColumn           ; 1行の文字数
                mov     cx, ax                  ;再描画する文字数
                VIDEO   0FFh
                ret
ScreenRewrite   endp


;*************************************************************************
;*                                                                       *
;*          仮想VRAMアドレスの計算(内部関数)                             *
;*          入力:       ax     = 行(1-  )                                *
;*                      bx     = 桁(1-  )                                *
;*          出力:       es:di = 仮想VRAMの位置                           *
;*                                                                       *
;*************************************************************************
_CalcVramAddr   proc    uses ax bx
                assume  ds:@data
                dec     ax                      ;行数補正
                mul     _ScreenColumn           ;横幅(文字数)
                shl     ax, 1                   ;横幅(バイト数)
                dec     bx                      ;桁数補正
                shl     bx, 1                   ;バイト数へ
                add     ax, bx                  ;仮想VRAM相対位置
                les     di, _VideoSegment       ;仮想VRAM先頭アドレス
                add     di, ax                  ; di = ((y-1)*画面幅+(x-1)*2)
                ret
_CalcVramAddr   endp

                end
```

を越えて正規化するような事態はありませんので，仮想VRAM上での相対位置を計算後，さきほど求めたDIの値を足し込むようにすれば問題ありません．この仮想VRAMは言葉どおり，仮想的なもので，このVRAMに直接書き込んでも，実際に画面表示が更新されるわけではありません．この仮想VRAMの内容を実際の画面に反映させるためには，リスト2.10やリスト2.11のような関数を使用して，画面の再描画を行います．

リスト2.10の関数では，更新する仮想VRAMの先頭アドレスと文字数を渡すことにより，その部分の画面だけを更新します．

リスト2.11の関数では，簡易的に開始行から終了行までをまとめて更新する形式をとっています．これは，将来的にテキストウィンドウなどを作成する場合，漢字の泣き別れなどが起きないように1行単位で更新を行っています．本来的には，必要な部分だけを更新したほうがパフォーマンス的にもよい場合があります．また，漢字のような2バイト文字は一度の画面更新で再表示させなければなりません．

ScreenRewrite関数で使用している，_CalcVramAddr関数は，行・桁位置から仮想VRAMでのアドレスを計算する関数で，内部的に使用しています．ただし，リスト2.1のGetVideoInfo関数およびリスト2.9のSetVramSegment関数実行時にセットした内部変数を使用していますので，ScreenRewrite（_CalcVramAddr）関数を使用する前にGetVideoInfo関数とSetVramSegment関数を呼び出してください．

DOS/Vの特徴はこの仮想VRAM方式にあるといっても過言ではなく，この仮想VRAMをじょうずに使いこなすのがこつです．たとえば，テキストウィンドウなどのちょっと複雑な画面は，とりあえず仮想VRAMに書き込んでおいて，一気に表示することが可能となっています．

また，実際に文字列を書き込むためには，リスト2.12に示す2関数があれば十分でしょう．ただし，このうちPutStringAbs2関数は仮想VRAMに書き込むだけで再描画はしていませんが，もうひとつのPutLoopChar関数は再描画しています．前者の関数でも，後者と同様に，DI,CXを保存しておいて，VIDEO 0FFhを呼び出せば，再描画することも可能です．

リスト2.12 文字列の表示2・文字の連続表示TEXTCTRL.ASM

```
include                 std.inc

                .code

;*********************************************************************
;*                                                                   *
;*        文字列の表示2(仮想VRAMに書き込むだけ)                      *
;*        void      PutStringAbs2(x,y,str,atr,cnt);                  *
;*        パラメータ： x ,y  表示開始座標                            *
;*                     str   文字列                                  *
;*                     atr   画面属性                                *
;*                     cnt   表示文字数                              *
;*        戻り値：     なし                                          *
;*                                                                   *
;*********************************************************************
PutStringAbs2   proc    uses ds es si di,¥
                                x:word, y:word, dstr:PSTR, atr:byte, cnt:word
                assume  ds:@data
                mov     ax, y                   ;行
                mov     bx, x                   ;桁
                call    _CalcVramAddr           ;仮想VRAMアドレスの計算
                if @model gt 3
                lds     si, dstr                ;文字列の先頭アドレス
                else
                mov     si, dstr                ;文字列の先頭アドレス
                endif
                mov     ah, atr                 ;表示属性
                mov     cx, cnt                 ;表示文字数
                .repeat
                    lodsb                       ;次の文字
                    stosw                       ;文字と属性の書き込み
                .untilcxz
                ret
PutStringAbs2   endp

;*********************************************************************
;*                                                                   *
;*        文字の連続表示                                             *
;*        void      PutLoopChar(x,y,chr,atr,cnt);                    *
;*        パラメータ： x ,y  表示開始座標                            *
;*                     chr   表示する文字                            *
;*                     atr   画面属性                                *
;*                     cnt   表示文字数                              *
;*        戻り値：     なし                                          *
;*                                                                   *
;*********************************************************************
PutLoopChar     proc    uses bx cx es di,¥
                                x:word, y:word, chr:byte ,atr:byte, cnt:word
                mov     ax,[y]                  ;行
                mov     bx,[x]                  ;桁
                call    _CalcVramAddr           ;仮想VRAMアドレスの計算

                mov     ah, atr                 ;表示属性
                mov     al, chr                 ;表示文字
                mov     cx, cnt                 ;表示文字数

                PUSHM   <di,cx>
                rep     stosw                   ;文字の仮想VRAMへのセット
                POPM    <cx,di>

                VIDEO   0FFh                    ;画面再描画
                ret
PutLoopChar     endp

                end
```

2-8 スクロール

DOS/Vへの批判として，スクロール速度が遅いことが，よくいわれています．ここでは，スクロール機能に関して解説します．

◎スクロール機能は，ビデオBIOSにスクロールアップ，スクロールダウン機能があります．

◎仮想VRAMでのデータのスクロールや画面の再描画を利用して，高速にスクロールさせることもできます．

◎PC-9801と異なり，グラフィック画面をスクロールさせるため，処理時間はかかります．しかし，ハードウェアスクロールやアクセレータ機能を使用したスクロールなどが使用され，実用上は問題ない速度でスクロールできます．

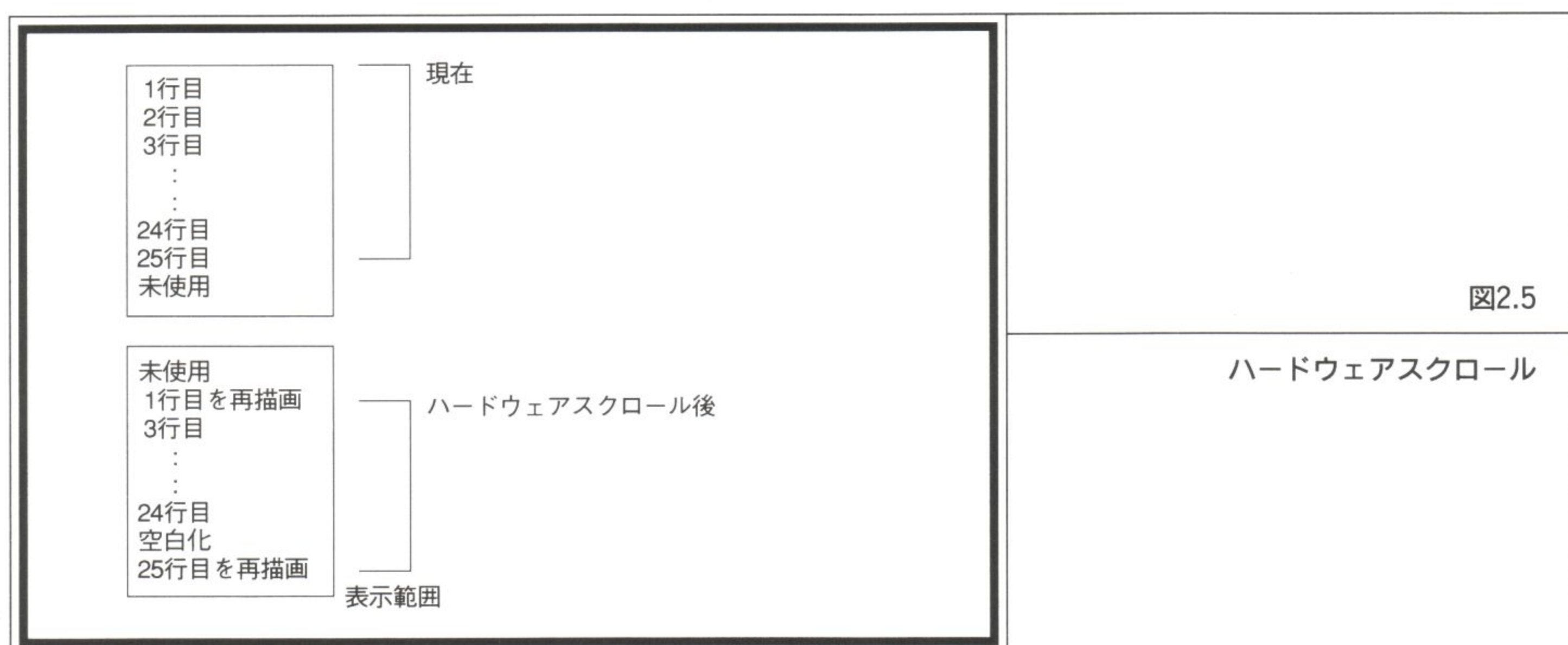
まず，スクロール機能に関する基礎知識をまとめておきましょう．

ハードウェアスクロール

$DISP.SYSには，“/HS=”というハードウェアスクロールを制御するオプションがあります．標準の$DISP.SYSでは，隠しオプション（ただし，雑誌やネットワークなどで，公にされていました）でしたが，DOS/V Extensionでは公開されています．

/HS=ON	ハードウェアスクロールを行う
/HS=LC	ハードウェアスクロールの変形
/HS=OFF	ハードウェアスクロールを行わない

1行目 2行目 3行目 : 24行目 25行目 未使用	現在	
未使用 1行目を再描画 3行目 : 24行目 空白化 25行目を再描画	ハードウェアスクロール後 表示範囲	

図2.5

ハードウェアスクロール

では，ここでいうハードウェアスクロールとは，一体何者なのでしょうか．さきほども説明したとおり，DOS/Vでは画面をグラフィック表示しています．そこで画面を直接スクロールさせると，PC-9801などのテキストスクロールと比較して遅くなります．そこで，CRTコントローラ（画面制御を行うLSIのことで，CRTCと略す）を制御してあたかもスクロールしたように見せているのが，ハードウェアスクロールです．

VGA用のCRTCには“Start Address”と呼ばれるレジスタがあり，グラフィックモード画面では，このレジスタに設定した値が指し示すビデオバッファのバイトアドレスから表示を行えます．この機能をうまく使用しているのが，DOS/Vのハードウェアスクロールです．表示開始アドレスを変更することによって，グラフィック画面をあたかもスクロールしているように見せかけています．

しかし，常識的にもわかるとおり，この手法は全画面スクロールにしか通用できないはずです．そこで$DISP.SYSでは，スクロール範囲をチェックしてハードウェアスクロールを使用したほうが速いと判断できる場合は，全画面をハードウェアスクロールさせておいて，本来移動しない部分を再描画しています．たとえば，1行目と25行目を残して，その間をスクロールさせる場合，図2.5のようになります．

この再描画が，DOS/Vでよくいわれていた画面の上下のちらつきの原因です．

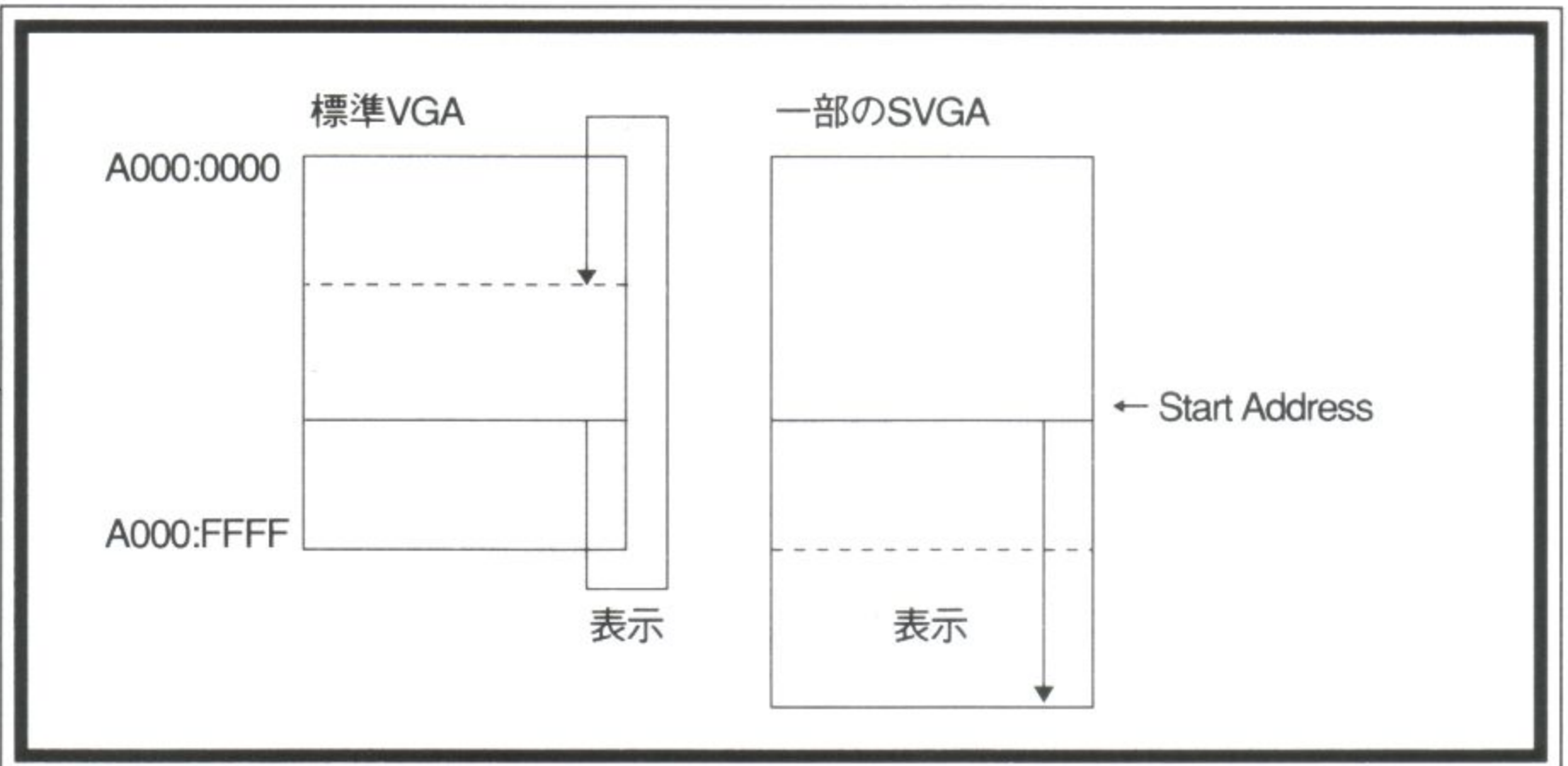
ラップラウンド処理

もうひとつの問題が，グラフィックVRAMのラップラウンド処理です．IBM PCの場合，グラフィックVRAMは，A000:0～A000:FFFFに存在していますが，ハードウェアスクロールを順次実行していくと，実VRAMがあふれてしまうのは，理解いただけるでしょう．

IBM純正のVGAは表示時にVRAMの終わりに達してしまい，オーバーフローした場合には，また，ラップラウンドにVRAMの先頭から表示するようになっています．

しかし，一部のSVGAチップ（たとえば，ET4000）では，このラップラウンド処理を行わないものがあります．この場合，$DISP.SYSはラップラウンド処理が行われているものだと思って，次のデータをVRAMの先頭から書き込んでしまうのですが，実際は，そのまま連続域が表示されてしまい，画面の下に白いゴミが表示されてしまいます．

図2.6

ラップラウンド処理

これを解決するのが，$DISP.SYSの/HS=LCオプションです．この/HS=LCとはいったいどういう意味なのでしょうか．HSは，当然“Hardware Scroll”の略であり，LCは実は“Line Compare”の略です．このLine CompareはさきほどのStart Addressと同じように，CRTCの1レジスタです．画面表示の走査線が，このレジスタにセットされた値と等しい回数表示されると，表示VRAM位置を記憶している内部レジスタをクリアして，VRAMの先頭より表示を行います．

前述のハードウェアスクロールのラップラウンド処理の話と比較してみるとわかると思いますが，一部のSVGAチップでは，ラップラウンド処理が必要な場合に，ラップラウンドしなくて済む走査線行数をこのLine Compareレジスタにセットすれば，そこで自動的に折り返してVRAMの先頭から表示してくれるのです．

よく，/HS=LCをつけるとソフトウェアスクロールになるので，非常に遅いといった話を聞きますが，実際はこのようにハードウェアスクロールを使用しています．ただし，画面表示を行っている最中は，CRTCがこのLine Compareレジスタの値を参照しているので，このレジスタの値を変更する場合は，垂直走査線復帰期間内にアクセスしなければなりません．これが/HS=LCの若干遅くなる理由です．

また，ごくまれに，Start Address機能自身も作動しないSVGAチップがありますが，この場合は，/HS=OFFをつければ，完全にソフトウェアでスクロールを行います．

スクロールプログラム

スクロール機能に関しては，ビデオBIOSにもありますが，いろいろテストした範囲では，仮想VRAMにアクセスできる場合は，仮想VRAM内でデータをスクロールさせてしまい，その範囲を再描画させたほうがチラツキが少なく，スムースにスクロールしているように見えます．リスト2.13に，仮想VRAM上のデータを直接スクロールさせる関数をのせておきます．この関数ではデータのスクロールだけを行い，実際に画面の再描画は行っていません．また，スクロールさせた残りの部分も，通常は別のデータを書き込むケースが多いので，そのまま残しています．

実はこの原稿を書いている最中に，Diamond STEALTH 24を手に入れてしまいました．このビデオカードはS3社の86C801を使用しているのですが，これとDISPS3.EXE V1.53の組み合せでスクロールのテストとして，V-Text（100×31）の画面でVZの画面を縦分割しスクロールさせて見たところ，実用上まったく問題ないスクロール速度を得ることができ

リスト2.13　仮想VRAM内データの上方向スクロールSCROLL.ASM

```
include                 std.inc

                .code

;****************************************************************************
;*                                                                          *
;*        仮想VRAM内データ・上方向スクロール                                  *
;*        void      ScrollUp(int x1, int y1, int x2, int y2, int line);     *
;*        入力:     x1   = 左上桁位置                                        *
;*                  y1   = 左上行位置                                        *
;*                  x2   = 右下桁位置                                        *
;*                  y2   = 右下行位置                                        *
;*                  line = スクロール行数                                    *
;*                                                                          *
;****************************************************************************
ScrollUp        proc    uses ds es si di,¥
                                x1:word, y1:word, x2:word, y2:word, line:word
                assume  ds:@data
                mov     ax,[y1]                 ;行
                mov     bx,[x1]                 ;桁
                call    CalcVramAddr            ;仮想VRAMアドレスの計算
                mov     si,di
                mov     ax,ScreenColumn         ; 1行の文字数
                shl     ax,1                    ; 1行のバイト数
                push    ax
                mov     cx,[line]               ;スクロール行数
                mul     cx
                add     si,ax                   ;スクロール前ポインタ
                pop     ax
                mov     dx,[x2]
                sub     dx,[x1]
                inc     dx                      ; dx =スクロール幅
                mov     cx,[y2]                 ;終了行数
                sub     cx,[y1]                 ;開始行数
                sub     cx,[line]               ;スクロール行数を減らす
                inc     cx                      ; cx =スクロール行数
                MOVSEG  ds,es
                .repeat                         ;行毎に複写
                    PUSHM   <cx,di,si>
                    mov     cx,dx               ;スクロールバイト幅数
                    rep     movsw               ; 1行スクロール
                    POPM    <si,di,cx>
                    add     si,ax               ; 1行分のバイト数を足す
                    add     di,ax
                .untilcxz
                ret
ScrollUp        endp

                end
```

リスト2.14　ビデオBIOSでのスクロール

```
include                 std.inc

                .code

;****************************************************************************
;*                                                                          *
;*        テキストVRAM・上方向スクロール                                      *
;*        void      ScrollUpBios(int x1, int y1, int x2, int y2, int line,  *
;*                          int attr);                                      *
;*        入力:     x1   = 左上桁位置                                        *
;*                  y1   = 左上行位置                                        *
;*                  x2   = 右下桁位置                                        *
;*                  y2   = 右下行位置                                        *
;*                  line = スクロール行数                                    *
;*                  attr = スクロール後の空白行の属性                         *
;*                                                                          *
;****************************************************************************
ScrollUpBios    proc    uses bx cx dx, x1:byte, y1:byte, x2:byte, y2:byte,¥
                                line:byte,attr:byte
                mov     ch, y1                  ; 行
                mov     cl, x1                  ; 桁
                sub     cx, 101h                ; 行桁調整
                mov     dh, y2                  ; 行
                mov     dl, x2                  ; 桁
                sub     dx, 101h                ; 行桁調整
                mov     bl, line                ; スクロールする行数
                mov     al, attr                ; スクロール後の空白行の属性
                VIDEO   06h
                ret
ScrollUpBios    endp

                end
```

ました．技術の進歩はすごいものだと思っており，これからは，そんなに工夫する必要性はないのかもしれません．

また，ビデオBIOSを使用する場合には，スクロールアップ（機能コードAH=06h）とスクロールダウン（機能コードAH=07h）がサポートされています．スクロールさせるためには，以下の呼び出しを使用します．

CX =	スクロール範囲の左上隅の行桁位置，画面の左上端が0
DX =	スクロール範囲の右下隅の行桁位置，画面の左上端が0
BH =	スクロール後の空白行の属性
AL =	スクロールさせる行数
VIDEO　06h	スクロールアップ
(VIDEO　07h	スクロールダウン)

サンプルプログラム（リスト2.14）は上方向のスクロールだけですが，スクロールダウンに関しては，06hのかわりに07hを指定してください．また，ALに0を指定すると，CX，DXで指定された矩形領域内がBHの属性で消去されます．

高速スクロールへの挑戦

何度もいうようですが，DOS/Vはいままで何度も雑誌に取り上げられているように，テキストモードの場合でも，最終的にはシステム（$DISP.SYSまたはV-Textドライバ）がグラフィック画面に文字を展開しています．したがって，ハードウェア的なテキストVRAMをもつ機種と比較すると，自由度は非常に高くなります（ここがすごく重要だと思っています）が，スクロール速度は状況によっては遅いものとなってしまいます．

今年（1993年）の2月ごろに話題となったNECのテレビコマーシャルでも，DOS/VマシンとNECの新製品のスクロール速度が比較されていました．しかし，本当にDOS/Vマシンはあれほどスクロールが遅いのだろうかという疑問があります．

そこで，DOS/Vでどのくらいの速度でスクロールが可能か，挑戦してみることにしました．

スクロール速度に関連するファクタとしては，以下の2種類があります．

(1)　キーボードのリピート速度

(2)　実際に画面をスクロールさせる時間

キーボードのリピート速度が遅ければ，手でカーソルキーを押し下げていれば遅くなるのはあたりまえでしょう．この件に関しては，「第３章　キーボード編」で解説します．

つぎに，実際に画面をスクロールさせる時間に関して検討してみましょう．前述しましたが，スクロールするには，ビデオBIOS機能（AH=06h,AH=07h）を使用する方法，仮想VRAM内のデータをスクロールさせる方法，すべて書き直してしまう方法の3種類があります．実際のコーディング方法を考えれば，ほかにもいろいろとはありますが．

今回テストしてみた範囲内では，仮想VRAM内のデータをスクロールさせ，最後にDOS/Vの画面再描画機能を使用する方法がもっとも速く，またスムースにスクロールしました（付録ディスクのサンプルプログラムSDP.EXEの場合）．ただし，スクロールさせる範囲やビデオドライバによってスクロール速度は異なりますので，ケースバイケースで判断してください．

ここでつぎの文章を読む前に，実際にDOS/Vの機種をお持ちの方は，添付のSDP.EXEを，まずは通常のテキストモードで実行してみてください．[↑]，[↓]キーで，スクロールします．押し続けると，中速スクロール，さらに，[Shift]＋[↑]，[Shift]＋[↓]キーで高速スクロールします．いろいろな機種がありますので結果は違うかもしれませんが，スクロールでおかしな点はあったでしょうか．

では，本題に戻りましょう．スクロールのさせ方にはもうひとつ，「目にも止まらないスクロール」というのがあります．本当に目にも止まらないスクロールであれば，人間の目ではなかなか追いつけないでしょう．そこで，1行スクロールを2行に，さらには4行にしてみました．実際に高速にスクロールをしている最中は，おおまかに文字のイメージがわかればよいわけで，目的の場所に近づいたら，1行ごとにスクロールすれば十分なはずです．

この検討結果から，スクロールを1回実行するたびに，キーボードからスクロール指示が継続してあるかをチェックし，継続してある場合にはスクロール行数を2行にしています．また，[Shift]キーが押されている場合には，スクロール行数を2倍にしています．これにより，実際のスクロール行数は1行から4行の間で変化します．単独で[↑][↓]キーを押せば1行ごとにスクロールしますから，問題はないはずです．

2.9 パレットの取得と設定

カラーパレット操作に関しては，ビデオシステムの進化にともなって拡張されてきており，CGAベースのJ3100，EGAベースのAX，VGAベースのDOS/Vと段々に機能が拡張されてきています．ここでは，カラーパレットの操作に関して解説します．

◎テキストモードでは同時に16色まで指定できます．また，オーバースキャンレジスタにより画面の外枠の色を指定できます．

◎実際には，64個のカラーレジスタ中の16個を指定して，表示カラーを選択します．

◎1個のカラーレジスタは3個の色要素から構成されていて，各色要素は00h～3Fh（64段階）の値で指定できるため，指定上は最大26万色まで可能です．ただし，実際にはビデオシステムにより最大色数は決まります．

◎PC-9801と異なり，カラーパレットは読み出し，書き込みが可能です．

DOS/Vのテキストモードは基本的に前景色16色，背景色16色です．この色は，画面モード設定時に省略時の値に設定され，ビデオBIOSの呼び出しにより変更することが可能です．実際に使用できる色は，16個のカラーパレットに設定されたカラーレジスタにより決定されます（表2.7）．すなわち，DOS/Vでは，同時発色は16色までですが，カラーパレット／カラーレジスタを操作することにより，いろいろな色を使用することができます．

また，カラーレジスタ自身は64個あり，この64個のカラーレジスタはそれぞれ，赤・緑・青の重みづけを00h～3Fh（64段階）の値で指定できますので，この値を変更して，色相をじょうずに影がついているように設定することにより，テキスト画面でも，疑似的な立体画面を作成したりすることができます（表2.8，2.9）．フリーソフトウェアのCAやJBOOTがこの機能を使用しています．

では，プログラミングを行う場合には，パレットの扱いをどのように考えたらよいのでしょうか．ひとつはまったく無視することです．とくに標準の色だけを使用し，プログラム起動時に画面モードを設定していて，かつ子プログラムを呼び出さない場合（呼び出しても戻ったときに画面モードを再設定している場合には問題ありません）には，無視して

表2.7　パレットの初期値

パレット番号	カラーレジスタ	表示色	パレット番号	カラーレジスタ	表示色
00	00	黒	08	38	灰色
01	01	青	09	39	薄い青
02	02	緑	0A	3A	薄い緑
03	03	水色	0B	3B	薄い水色
04	04	赤	0C	3C	薄い赤
05	05	紫	0D	3D	薄い紫
06	14	茶色	0E	3E	薄い黄
07	07	白	0F	3F	明るい白

※表示はすべて16進数表示

表2.8 カラーレジスタの初期値

番号	赤	緑	青	表示色	番号	赤	緑	青	表示色
00	00	00	00	黒	20	15	00	00	暗い赤
01	00	00	2A	青	21	15	00	2A	
02	00	2A	00	緑	22	15	2A	00	黄緑
03	00	2A	2A	水色	23	15	2A	2A	
04	2A	00	00	赤	24	3F	00	00	明るい赤
05	2A	00	2A	紫	25	3F	00	2A	濃いピンク
06	2A	2A	00	黄	26	3F	2A	00	やまぶき
07	2A	2A	2A	白	27	3F	2A	2A	
08	00	00	15	暗い青	28	15	00	15	暗い紫
09	00	00	3F	明るい青	29	15	00	3F	すみれ
0A	00	2A	15		2A	15	2A	15	
0B	00	2A	3F	緑青	2B	15	2A	3F	
0C	2A	00	15		2C	3F	00	15	紫赤
0D	2A	00	3F		2D	3F	00	3F	明るい紫
0E	2A	2A	15		2E	3F	2A	15	
0F	2A	2A	3F		2F	3F	2A	3F	
10	00	15	00	暗い緑	30	15	15	00	暗い黄色
11	00	15	2A		31	15	15	2A	
12	00	3F	00	明るい緑	32	15	3F	00	
13	00	3F	2A	空色	33	15	3F	2A	
14	2A	15	00	茶色	34	3F	15	00	
15	2A	15	2A		35	3F	15	2A	
16	2A	3F	00		36	3F	3F	00	明るい黄色
17	2A	3F	2A		37	3F	3F	2A	
18	00	15	15	暗い水色	38	15	15	15	灰色
19	00	15	3F		39	15	15	3F	薄い青
1A	00	3F	15	青緑	3A	15	3F	15	薄い緑
1B	00	3F	3F	明るい水色	3B	15	3F	3F	薄い水色
1C	2A	15	15		3C	3F	15	15	薄い赤
1D	2A	15	3F		3D	3F	15	3F	薄い紫
1E	2A	3F	15		3E	3F	3F	15	薄い黄
1F	2A	3F	3F		3F	3F	3F	3F	明るい白

※表示はすべて16進数表示

もかまわないと思います．しかし，色をじょうずに使用することで，画面の強弱をつけている場合などは，カラーパレットやカラーレジスタを操作する必要があります．また，子プログラムを呼び出す場合なども，自分で画面モードを元に戻すか，またはカラーレジスタ／カラーパレットを元の値に戻したください．

カラーパレット／カラーレジスタの設定には，AH=10hのビデオBIOSを使用します．ただし，AH=10hにはサブ機能コードをALで指定する必要があります．

個別パレット ――――――――――――――――――――――

BL =　　　　　読み取るパレットレジスタ番号（0-15）
VIDEO　1007h パレットレジスタの個別読み取り
[戻り値]
BH =　　　　　指定したパレットレジスタに設定されていたカラーレジスタ番号

BL =　　　　　設定するパレットレジスタ番号（0-15）
BH =　　　　　パレットレジスタに設定するカラーレジスタ番号
VIDEO　1000h パレットレジスタの設定

一括パレット ――――――――――――――――――――――

ES:DX =　　　パレットレジスタ（0-15）とオーバースキャンレジスタの現在の値を保管する17バイトの記憶域先頭アドレス

表2.9 カラーレジスタの初期値（色系列別）

番号	赤	緑	青	表示色	番号	赤	緑	青	表示色
00	00	00	00	黒	30	15	15	00	暗い黄色
38	15	15	15	灰色	06	2A	2A	00	黄
07	2A	2A	2A	白	0E	2A	2A	15	
3F	3F	3F	3F	明るい白	36	3F	3F	00	明るい黄色
					3E	3F	3F	15	薄い黄
08	00	00	15	暗い青	37	3F	3F	2A	
01	00	00	2A	青					
31	15	15	2A		0A	00	2A	15	
09	00	00	3F	明るい青	1A	00	3F	15	青緑
39	15	15	3F	薄い青	13	00	3F	2A	空色
0F	2A	2A	3F		33	15	3F	2A	
10	00	15	00	暗い緑	11	00	15	2A	
02	00	2A	00	緑	19	00	15	3F	
12	00	3F	00	明るい緑	0B	00	2A	3F	緑青
2A	15	2A	15		2B	15	2A	3F	
3A	15	3F	15	薄い緑					
17	2A	3F	2A		21	15	00	2A	
					29	15	0	03F	すみれ
18	00	15	15	暗い水色	0D	2A	00	3F	
03	00	2A	2A	水色	1D	2A	15	3F	
23	15	2A	2A						
1B	00	3F	3F	明るい水色	0C	2A	00	15	
3B	15	3F	3F	薄い水色	2C	3F	00	15	紫赤
1F	2A	3F	3F		25	3F	00	2A	濃いピンク
					35	3F	15	2A	
20	15	00	00	暗い赤					
04	2A	00	00	赤	22	15	2A	00	黄緑
1C	2A	15	15		32	15	3F	00	
24	3F	00	00	明るい赤	16	2A	3F	00	
3C	3F	15	15	薄い赤	1E	2A	3F	15	
27	3F	2A	2A						
					14	2A	15	00	茶色
28	15	00	15	暗い紫	34	3F	15	00	
05	2A	00	2A	紫	26	3F	2A	00	やまぶき
15	2A	15	2A		2E	3F	2A	15	
3D	3F	15	3F	薄い紫					
2D	3F	00	3F	明るい紫					
2F	3F	2A	3F						

※表示はすべて16進数表示

VIDEO　1009h パレットレジスタの一括読み取り

ES:DX =　パレットレジスタ（0-15）とオーバースキャンレジスタに設定する値がセットされている17バイトの記憶域先頭アドレス

VIDEO　1002h パレットレジスタの一括設定

オーバースキャンレジスタ ――――――――――――――――――

VIDEO　1008h オーバースキャンレジスタの読み取り

[戻り値]

BH =　オーバースキャンレジスタに設定されていたカラーレジスタ番号

BH =　オーバースキャンレジスタに設定するカラーレジスタ番号

VIDEO　1001h オーバースキャンレジスタの設定（画面の外枠の色）

個別カラーレジスタ ――――――――――――――――――――

BX=　読み取るカラーレジスタ番号

VIDEO　1015h カラーレジスタの個別読み取り

[戻り値]

DH =　　　読み取った赤色の値
CH =　　　読み取った緑色の値
CL =　　　読み取った青色の値

BX =　　　設定するカラーレジスタ番号
DH =　　　設定する赤色の値
CH =　　　設定する緑色の値
CL =　　　設定する青色の値
VIDEO　1010h カラーレジスタの個別設定

一括カラーレジスタ ---------------------------------------

ES:DX =　　　カラーレジスタの現在の値を保管するテーブル先頭アドレス
　　　　　　テーブル形式：赤，緑，青，赤，緑，青，……
BX =　　　読み取る最初のカラーレジスタ番号
CX =　　　読み取るカラーレジスタの数
VIDEO　1017h カラーレジスタの一括読み取り

ES:DX =　　　カラーレジスタに設定する値がセットされているテーブル先頭アドレス
　　　　　　テーブル形式：赤，緑，青，赤，緑，青，……
BX =　　　設定する最初のカラーレジスタ番号
CX =　　　設定するカラーレジスタの数
VIDEO　1012h カラーレジスタの一括設定

リスト2.15に，パレットおよびカラーレジスタの一括保管，一括復元関数をのせておき

リスト2.15　　**パレットの保管・復元**

```
include         std.inc

                .code

;*******************************************************************************
;*                                                                             *
;*        パレットの保管                                                         *
;*        void        SavePalette(int kind, char *PaletteBuffer)               *
;*        パラメータ:kind           = 0 : パレットとカラーレジスタを保管          *
;*                                    1 :パレットだけを保管                     *
;*                    PaletteBuffeur = パレットとカラーレジスタを保管する        *
;*                                   バッファ.                                  *
;*                                    kind = 0の場合は209バイト必要.            *
;*                                       先頭 17バイトがパレット,               *
;*                                       次の192バイトがカラーレジスタ          *
;*                                    kind = 1の場合は192バイト必要.            *
;*                                                                             *
;*        パレットの復元                                                         *
;*        void        RestorePalette(int kind, char * PaletteBuffer)           *
;*        パラメータ:kind           = 0 : パレットとカラーレジスタを復元          *
;*                                    1 :パレットだけを復元                     *
;*                    PaletteBuffeur = パレットとカラーレジスタを設定する        *
;*                                   バッファ.                                  *
;*                                    kind = 0の場合は209バイト必要.            *
;*                                    kind = 1の場合は192バイト必要.            *
;*                                                                             *
;*******************************************************************************
SavePalette     proc    uses bx cx dx, kind:word, PaletteBuffer:PBYTE
                assume  ds:@data
                if @model gt 3                  ; large modelの場合
                les     dx, PaletteBuffer       ;パレット保存変数アドレス
                else
                mov     dx, PaletteBuffer       ; パレット保存変数のオフセット
                MOVSEG  es, ds                  ; パレット保存変数のセグメント
                endif
                VIDEO   1009h                   ;パレットの一括読み取り

                .if kind != 0
                    add     dx, 17              ;カラーレジスタ保存変数のオフセット
                    xor     bx, bx              ;カラーレジスタ0番より
                    mov     cx, 64              ;すべて (64個) を読み取る
                    VIDEO   1017h               ;カラーレジスタの一括読み取り
                .endif
                ret
SavePalette     endp


RestorePalette  proc    uses bx cx dx, kind:word, PaletteBuffer:PBYTE
                assume  ds:@data
                if @model gt 3                  ; large model の場合
                les     dx, PaletteBuffer       ; パレット保存変数アドレス
                else
                mov     dx, PaletteBuffer       ; パレット保存変数のオフセット
                MOVSEG  es, ds                  ; パレット保存変数のセグメント
                endif
                .if kind != 0
                    push    dx
                    add     dx, 17              ;カラーレジスタ保存変数のオフセット
                    xor     bx, bx              ;カラーレジスタ0番より
                    mov     cx, 64              ;すべて (64個) を設定する
                    VIDEO   1012h               ;カラーレジスタの一括設定
                    pop     dx
                 endif

                VIDEO   1002h                   ;パレットの一括設定
                ret
RestorePalette  endp

                end
```

ます．この関数を使用すれば，すべてのパレットおよびカラーレジスタを保管復元しますので，パレットやカラーレジスタを操作するプログラムの場合，プログラム起動時にSavePalette関数を，プログラム終了時にRestorePalette関数を呼び出すようにすれば，プログラム途中での保管復元は不要となります．

2.10 フォントの取得と設定

DOS/Vでは半角文字も全角文字もすべてソフトウェアで処理しているため，どちらの文字も同じように登録することができます．ここでは，フォントの登録・読み取りに関して解説します．

◎DOS/VのBIOSインターフェース解説書（『IBM DOSバージョンJ5.0/V BIOSインターフェース技術解説書』）では，16ドットと24ドットフォントに関してだけ解説されていますが，V-Text環境では，他のフォントサイズまで考慮しなければなりません．

◎フォントの登録，読み取りに関しては，ビデオBIOSにその機能があります．また，一部の機能はシステムBIOS（INT 15h）にあります．

◎Super Driversでは，フォント管理機能が強化されています．

DOS/Vで使用される半角フォントは図2.7，2.8のとおりです．とくに注意しなければいけないことは，半角罫線文字や矢印記号などよく使用されるものが異なっていることです．また，IBM PC系の汎用プログラムを作成する場合には，リスト1.1（P.15）の動作機種判定の戻り値を参照して，罫線コードなどを変えなければいけないことです．DOS/Vの日本語モードに限定すると，“↓”が文字コード07hに割り当てられているため，使用するビデオBIOSによってはベルコードとして取り扱われ，ブザーが鳴ってしまうことがありますので，仮想VRAM直書きを行うかまたはAX=132xh系の文字列の書き込みを使用してください．

全角文字に関しては，シフトJISコード（JIS 83年版に準拠）体系をとっていますが，一部IBM独自な部分があります．この点に関しては，第０章の「0.4　PC-9801との違い」（P.7）を参照してください．また，Super Driversに含まれるFONTEXやフリーソフトウェアのFONTXを使用している場合には，文字フォントを，自由に設定できますので，PC-9801やJ3100などと同じフォント体系を使用できる場合もあります．また，部分的に，PC-9801の丸数字などを組み込むこともできます．

フォントの読み取り，設定に関しては，DOS/Vでは半角文字も全角文字もすべてソフトウェアで処理しているため，どちらの文字も同じように登録することができます．このフォントの登録機能のうち，半角文字の生成機能は，パレットと同様に画面モードを設定す

図2.7

日本語モード時の半角フォント

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		┼		0	@	P	‘	p		|		ー	タ	ミ	↑	|
1	┌		!	1	A	Q	a	q	↑	|	。	ア	チ	ム	|	|
2	┐	↕	”	2	B	R	b	r	|	|	「	イ	ツ	メ	|	|
3	└		#	3	C	S	c	s	2	1	」	ウ	テ	モ	2	1
4	┘	■	$	4	D	T	d	t	バ	バ	、	エ	ト	ヤ	バ	バ
5	│	┴	%	5	E	U	e	u	イ	イ	・	オ	ナ	ユ	イ	イ
6	─	┬	&	6	F	V	f	v	ト	ト	ヲ	カ	ニ	ヨ	ト	ト
7	↓	┤	’	7	G	W	g	w	コ	目	ァ	キ	ヌ	ラ	コ	目
8			(	8	H	X	h	x	ー	|	ィ	ク	ネ	リ	ー	|
9	○	├	)	9	I	Y	i	y	ド	|	ゥ	ケ	ノ	ル	ド	|
A		■	*	:	J	Z	j	z	の	|	ェ	コ	ハ	レ	の	|
B	⊠	↵	+	;	K	[	k	{	|	|	ォ	サ	ヒ	ロ	|	|
C		↑	,	<	L	¥	l	|	|	|	ャ	シ	フ	ワ	|	↓
D		|	-	=	M	]	m	}	|	|	ュ	ス	ヘ	ン	|	
E	■	→	.	>	N	^	n	‾	|	|	ョ	セ	ホ	゛	|	
F	☼	←	/	?	O	_	o		|	↓	ッ	ソ	マ	゜	|	

図2.8

英語モード時の半角フォント

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		►		0	@	P		p	Ç	É	á	░	└	╨	α	≡
1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	☻	↕	”	2	B	R	b	r	é	Æ	ó	▓	┬	╥	γ	≥
3	♥	‼	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	μ	÷
7	•	↨	’	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	∙
A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
C	♀	∟	,	<	L	¥	l	|	î	£	¼	╝	╠	▄	∞	ⁿ
D	♪	↔	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	²
E	♫	▲	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	∈	■
F	☼	▼	/	?	O	_	o	△	Å	ƒ	»	┐	╧	▀	∩	

るまで有効です．すなわち，画面モードを設定するとフォントも最初のフォントにリセットされてしまいます．ただし，V-Textドライバによっては，リセットしないものもありますので，フォントを変更する前に，現状のフォントを保管してから，変更を行い，子プログラム呼び出し時および終了時にフォントを元に戻しておくほうが安全です．

また，フォントを処理する場合には，フォントのサイズに注意してください．現在でもフォントの高さを見ると，12，16，19，24ドットフォントがあります．また，標準のBIOS機能ではフォントの幅がわからないため，フォントの高さから推定するしかありません．

現状では，

フォントの高さ12 → 12(6) フォントの幅
16 → 16(8)
19 → 16(8)
24 → 24(12)

となっています．しかし，IBM DOS/V Extensionでは，14，18，20，26，29，30ドットなどの多彩なフォントの高さをもっており，かつ，フォントの高さからフォントの幅は一意に決めることはできません．これに対しては，後述のV-Text APIを使用することによ

り，フォントのサイズを取得しなければなりません．Super DriversやDISPS3などでも，V-Text APIの一部をサポートしていますので，今後はV-Text　APIを使用されることをお薦めします．

また，DOS/V Extensionでは，文字フォントはすべて16ドットベースで考えられており，表示ドライバが，自動的にフォントの圧縮・拡張を行っていますので，それほどフォントサイズに注意することはないと思われます．

文字フォントの取得と設定のためのビデオBIOSには，機能コード（AH=11h）の文字の生成と，機能コード（AH=18h）の文字フォントパターンの要求の2種類があります．

文字フォントパターンの取得 ------------------------------

BH = 00h　　予約済み
BL = 00h　　文字セット0（これ以外は予約済み）
CH = 　　全角文字の場合，シフトJISコードの1バイト目
　　半角文字の場合，00h
CL = 　　全角文字の場合，シフトJISコードの2バイト目
　　半角文字の場合，文字コード
DX = 　　フォントサイズ（DH = 文字のドット幅，DL = 文字のドット高さ）
　　060Ch　6×12ドット半角文字パターン（Super Drivers）
　　0810h　8×16ドット半角文字パターン
　　0813h　8×19ドット半角文字パターン
　　0C0Ch　12×12ドット全角文字パターン（Super Drivers）
　　1010h　16×16ドット全角文字パターン
　　1818h　24×24ドット全角文字パターン
ES:SI = 　　文字パターンを格納するバッファの先頭アドレス
VIDEO　1800h システムフォントバッファから文字パターンを読み取る
[戻り値]
AL = 　　00h　　正常終了
　　00h以外　エラー

全角文字フォントパターンの設定 --------------------------

同様に，文字フォントパターンを設定する場合には，上記，文字フォントパターンの読み取りと同様に，以下のビデオBIOSを使用しますが，この機能で登録できるのはマニュアルの表記ではわかりにくいのですが，全角文字だけですので注意してください．

BH = 00h　　予約済み
BL = 00h　　文字セット0（これ以外は予約済み）
CH = 　　シフトJISコードの1バイト目
CL = 　　シフトJISコードの2バイト目
DX = 　　フォントサイズ（DH = 文字のドット幅，DL = 文字のドット高さ）
　　0C0Ch　12×12ドット全角文字パターン（Super Drivers）
　　1010h　16×16ドット全角文字パターン

1818h　24×24ドット全角文字パターン

ES:SI =　　　文字パターンを格納しているバッファの先頭アドレス

VIDEO 1801h　システムフォントバッファへ文字パターンの書き込み

[戻り値]

AL =　　　00h　　正常終了

　　　　　00h以外　エラー

ただし，標準の$FONT.SYSでは，この機能を使用してで登録できるのは，全角のユーザー定義文字だけです．FONTEX.EXEおよびFONTX.SYSでは，すべての全角文字を登録できます．

またこの機能は，内部的にフォントの読み取りと書き込み機能（AH=50h,INT 15h）を呼び出していますので，高速な処理を必要とする場合は上記機能を使用してください．

半角文字フォントパターンの登録 ----------------------------

半角フォントを登録する場合には，機能コード（AH=11h）文字の生成を使用します．

ES:BP =　　　文字パターンを格納しているバッファの先頭アドレス

CX =　　　登録する文字数

DX =　　　テーブル内の最初の文字の文字コード（DHはつねに0）

BL = 00h　　文字ブロック

BH =　　　1文字当たりのバイト数

VIDEO　1100h ユーザーの1バイト文字セットの登録

BHには，1文字当たりのバイト数を設定しますが，フォントサイズから計算すると以下のとおりになります．

6×12 = 12バイト（1バイトの先頭6ビットを使用）

8×16 = 16バイト

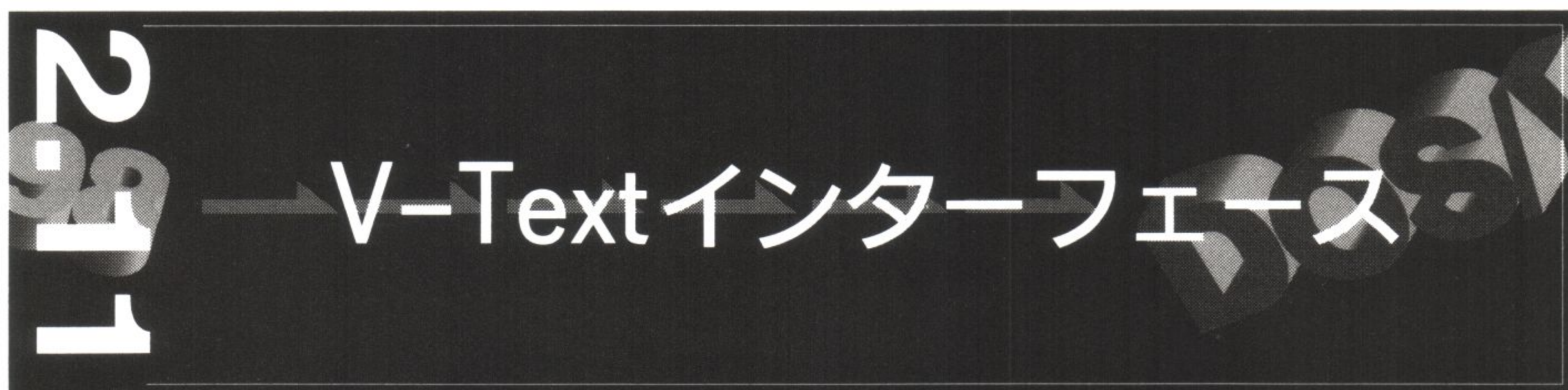
リスト2.16　　文字フォントパターンの取得と設定FONT.ASM

```
include         std.inc

                .code

;*********************************************************************
;*                                                                   *
;*       文字フォントパターンの取得                                    *
;*       int       GetFont(unsigned Code, unsigned FontSize,         *
;*                         char far *FonteBuffer)                    *
;*       パラメータ:Code          = 文字コード（全角,半角）            *
;*                 FontSize      = フォントサイズ                     *
;*                 FontBuffeur   = 文字パターンを格納するバッファ      *
;*                                                                   *
;*********************************************************************
GetFont         proc    uses bx cx dx es si,¥
                        Code:word, FontSize:word, FontBuffer:far ptr byte
                assume  ds:@data
                mov     cx, Code                ;文字コード
                mov     dx, FontSize            ;フォントサイズ
                les     si, FontBuffer          ;文字パターンを格納するバッファの
                                                ;先頭アドレス
                xor     bx, bx
                VIDEO   1800h                   ;文字パターンの読み取り
                ret
GetFont         endp

;*********************************************************************
;*                                                                   *
;*       文字フォントパターンの設定                                    *
;*       int       SetFont(unsigned Code, unsigned FontSize,         *
;*                         char far *FonteBuffer)                    *
;*       パラメータ:Code          = 文字コード（全角,半角）            *
;*                 FontSize      = フォントサイズ                     *
;*                 FontBuffeur   = 文字パターンを格納するバッファ      *
;*                                                                   *
;*********************************************************************
SetFont         proc    uses bx cx dx es si,¥
                        Code:word, FontSize:word, FontBuffer:far ptr byte
                assume  ds:@data
                mov     cx, Code                ;文字コード
                mov     dx, FontSize            ;フォントサイズ
                les     si, FontBuffer          ;文字パターンを格納するバッファの
                                                ;先頭アドレス
                xor     bx, bx
                VIDEO   1801h                   ;文字パターンの読み取り
                ret
SetFont         endp

;*********************************************************************
;*                                                                   *
;*       半角文字フォントパターンの設定                                *
;*       int       SetFontHalf(unsigned Code, unsigned FontCount,    *
;*                         unsigend FontSize, char far *FonteBuffer) *
;*       パラメータ:Code          = 文字コード                         *
;*                 FontCount     = 登録する文字数                     *
;*                 FontSize      = フォントサイズ                     *
;*                 FontBuffeur   = 文字パターンを格納するバッファ      *
;*                                                                   *
;*********************************************************************
SetFontHalf     proc    uses bx cx dx es,Code:word, FontCount:word,¥
                                FontSize:byte, FontBuffer:far ptr byte
                assume  ds:@data
                mov     dx, Code                ;文字コード
                mov     bh, FontSize            ;フォントサイズ
                mov     bh, FontSize            ;フォントサイズ
                les     bp, FontBuffer          ;文字パターンを格納するバッファの
                                                ;先頭アドレス
                xor     bl, bl
                VIDEO   1801h                   ;文字パターンの読み取り
                ret
SetFontHaif     endp

                end
```

8×19 = 19バイト

12×24 = 48バイト（1ラインにつき2バイトで先頭12ビットを使用）

リスト2.16のフォント登録とカラーパレット操作により，テキスト画面でありながら，一見グラフィックのような立体的な画面を作成することができます．

2.11 V-Textインターフェース

V-Textに関するAPI（Application Program Interface）は，フォント関連と表示関連の2種類があります．ここでは，代表的なV-Textドライバである，Super Driversのフォントドライバに関するAPIと，DOS/V Extension Ver.1.0のディスプレイドライバに関するAPIを解説します．

◎本来，Super DriversとDOS/V Extensionは同じものを目指しているのですが，実現方法には若干の相違点があります．

◎Super Driversのほうは，フォント関連のAPIを強化しているのに対して，DOS/V Exensionのほうはディスプレイ関連のAPIを強化しています．

◎DOS/V Exitensionのビデオモードの考え方はモード3でも高密度を実現するなど，多少異なった部分があります．

各V-Text対応ドライバにより，実装されているAPI機能が若干異なります．そのため，まず，どのV-Textドライバ（フォントおよびディスプレイドライバを別個に判断します）が使用されているかを判断しなけれななりません．

Super Drivers

Super Driversは，フォントドライバを制御するために，次のようなサービスを提供しています．

(1)　フォント読み取りプログラムのアドレスを得る．（DOS/V互換）

(2)　フォント書き込みプログラムのアドレスを得る．（DOS/V互換）

(3)　フォントドライバのバージョンを得る．

(4)　現在登録されているフォントの数を得る．

(5)　現在登録されているフォントの情報を得る．

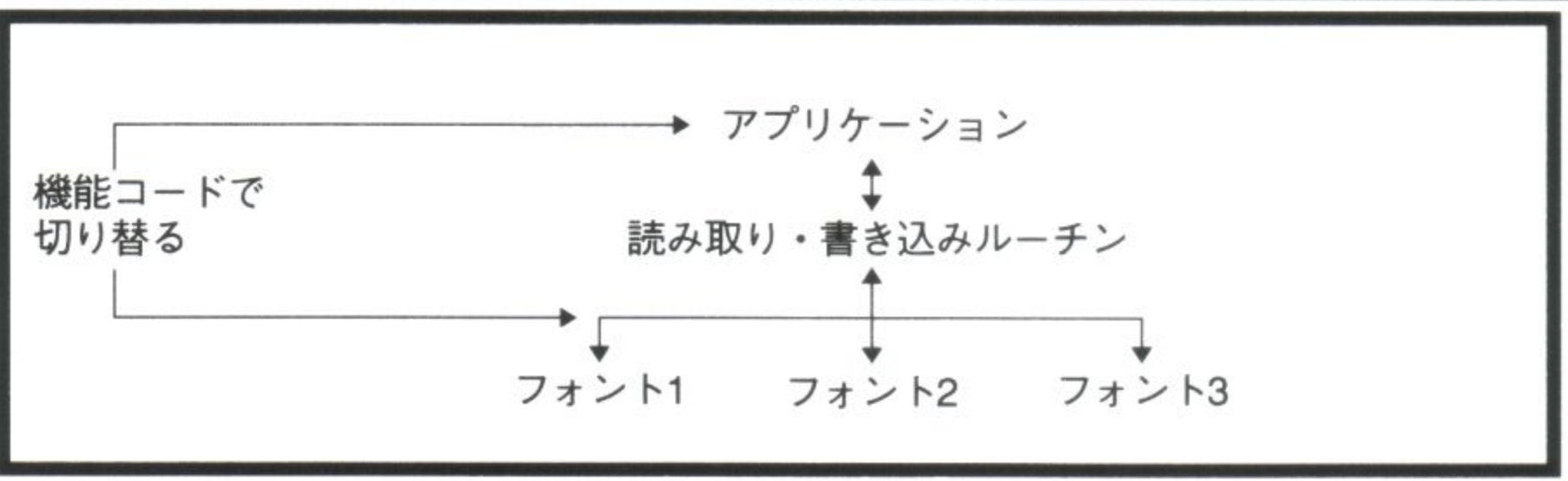
(6)　特定のフォントサイズに対して，現在選択されているフォント名を得る．

(7)　特定のフォントサイズに対して，現在選択されているフォントを変更する．

マルチフォント環境における注意点 ―――――――――――――

上記の(1)，(2)は，1つのサイズに対して1つのエントリが与えられます．そのため，プロ

図2.9

フォント読み取り／書き込み

グラムより同一サイズの複数フォントを読み取る場合は，機能コード7を利用して，フォントを選択をしながら読み取らなければなりません（図2.9）．

FONTEXのAPI機能アドレスの取得 ------------------------

(1) $IBMAFNTをint 21h AX=3D00hでオープンする．

(2) int 21h AX=4402hでエントリーアドレスを取得する．

(3) オープンしたデバイスをint 21h AH=3Ehでクローズする．

リスト2.17　FONTEXのAPIアドレスの取得FONT.ASM

```
include         std.inc

                .code

;*****************************************************************************
;*                                                                           *
;*          FONTEXのAPIアドレスの取得                                         *
;*          far char   *GetFontexApiEntry(void);                             *
;*          戻り値:    0       取得できなかった                               *
;*                     0以外   取得したAPIへのfarポインタ                     *
;*                                                                           *
;*****************************************************************************
FontexDeviceName    db      '$IBMAFNT',0        ; フォントドライバのデバイス名
FontexAPIEntry      dd      0                   ; FONTEX APIのエントリアドレス

GetFontexApiEntry   proc    uses ds
                MOVSEG  ds, cs
                assume  ds:@code

                mov     dx,offset cs:FontexDeviceName
                MSDOS   3D00h                   ;デバイスのオープン
                .if !carry?
                    mov     bx,ax

                    mov     dx,offset cs:FontexAPIEntry
                    mov     cx,4
                    MSDOS   4402h               ; IOCTL読み込み
                    pushf
                    MSDOS   3Eh                 ;デバイスのクローズ
                    popf
                    .if !carry?
                        mov     ax,word ptr FontexAPIEntry
                        mov     dx,word ptr FontexAPIEntry+2
                    .else
                        xor     dx, dx
                        xor     ax, ax
                    .endif
                .endif
                ret
GetFontexApiEntry   endp

                end
```

FONTEXには，以下のAPI機能があります．

(1) 現在登録されているフォントの数を得る（AX=50F2h）

(2) 現在登録されているフォントの情報を得る（AX=50F3h）

(3) 現在選択されているフォントの名前を得る（AX=50F4h）

(4) 新しくフォントを選択する（AX=50F5h）

現在登録されているフォントの数を得る --------------------

AX = 50F2h

CALL　　FontexApiEntry

[戻り値]

AH = 　　　　エラーコード（0なら正常，それ以外ならエラー）

AL = 　　　　登録されているフォントの数

現在登録されているフォントの情報を得る ------------------

ES:BX = 　　テーブルを格納するアドレス．機能コードF2hであらかじめ数を得て，フォントの種類×11バイト分のメモリを確保しておかなければなりません．

AX = 50F3h

CALL　FontexApiEntry

[戻り値]

AH =　エラーコード（0なら正常，それ以外ならエラー）

ES:BXで示されるアドレスからテーブルが格納されています．テーブルの形式は，

1バイト: XSize（フォントの横幅）

1バイト: YSize（フォントの縦幅）

1バイト: CodeType（フォントの種類0=ASCII,1=SJIS）

8バイト: FontName（フォントの名前 8 文字）

であり，これがフォントの数だけ並んでいます．

現在選択されているフォントの名前を得る ------------------

BH =　フォントの種類（ビット0=0：1バイト文字，ビット0=1：2バイト文字）

DH =　文字幅

DL =　文字の高さ

BP =　コードページ（=0）

ES:DI =　フォントの名前を格納するアドレス（8バイトの領域が必要です）

AX = 50F4h

CALL　FontexApiEntry

[戻り値]

AH =　エラーコード（0なら正常，それ以外なら　エラー）

ES:DIで示されるアドレスに，フォントの名前が格納されます．

新しくフォントを選択する ---------------------------------

BH =　フォントの種類（ビット0=0：1バイト文字,ビット0=12バイト文字）

DH =　文字幅

DL =　文字の高さ

BP =　コードページ（=0）

ES:DI =　フォントの名前が入っているアドレス

AX = 50F5h

CALL　FontexApiEntry

[戻り値]

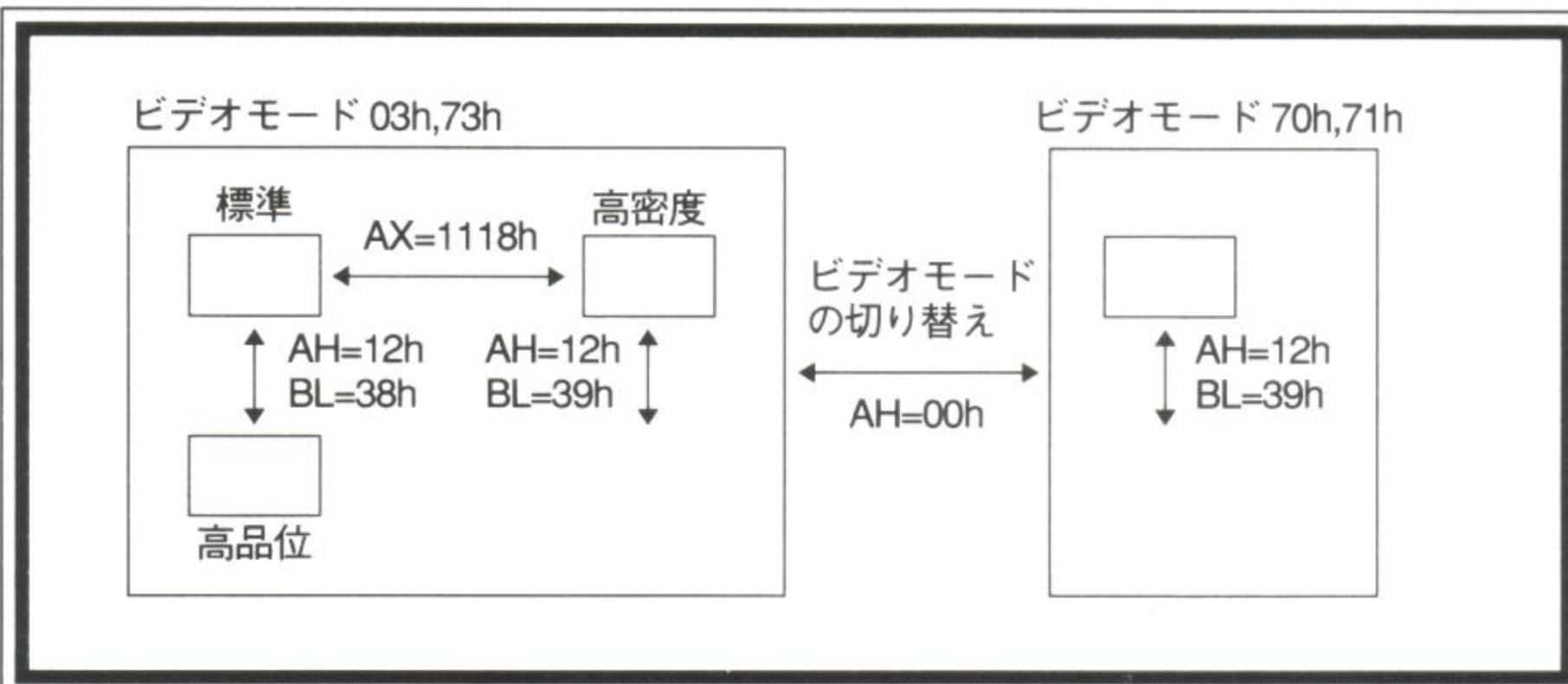
AH =　エラーコード（0なら正常，それ以外ならエラー）

DOS/V Extension Ver.1.0

DOS/V Extensionの大きな違いは，ビデオシステムとビデオドライバが1対1で，ビデオモードの考え方が異なることです．たとえば，Super Driversでは，同じビデオシステムでも解像度が異なれば別のドライバを指定します．実際のデバイスドライバはDISPEXで共通ですが，そこで指定するドライバ名が異なります．しかし，DOS/V Extensionでは同じドライバで複数の解像度をサポートしています．したがって，それらの解像度（フォ

図2.10

DOS/V Extinsionの画面モードの関連

ビデオモード03h,73h
標準
AX=1118h
高密度
AH=12h
BL=38h
AH=12h
BL=39h
高品位
ビデオモード
の切り替え
AH=00h
ビデオモード70h,71h
AH=12h
BL=39h

ントの品位と密度）の切り替えのために，別途いくつかのAPIが提供されています．

基本的な考え方としては，ビデオモード03h,73hには標準モード（16ドットフォント）と高品位モード（24ドットフォント）があります．この高品位モードは，XGA系とET4000系とDA2系だけで，通常のプログラムに対してトランスペアレントになっています（フォント関連の機能も内部的に16ドットを24ドットに拡張しています）．すなわち，高品位モードに対応していないプログラムでも，基本的にはそのまま24ドットフォントの画面表示で動作させることができます．また，別の観点で，標準モード（80桁×25行）と高密度モード（80桁×26行以上）があります．すなわち，画面幅が80桁のままであれば，ビデオモードが03h，73hのままで高密度に切り替えることもできます．さらに，画面幅を81桁以上に拡張したのが，ビデオモード70h，71hです（例外的に，VGA用のドライバでは，画面幅が80桁のままのビデオモード70h，71hがあります）．

マニュアル上はどちらも高密度モードと呼ばれていますが，ここではわかりやすくするため，ビデオモード70h，71hのほうを可変高密度モードと呼びます．別の呼び方をすると，標準（80桁×25行），縦長（80桁×26行以上），ワイド（81桁以上）の3段階に別れ，縦長やワイドには複数の密度が存在する場合があります（図2.10）．

ビデオモードを切り替えた場合，どの画面（品位・密度）が出てくるかは，デフォルトの品位・密度の定義にしたがって行われます．このデフォルトの定義は，DSPX.PRO中に記録されています．そして，それらを切り替えるためにサポートされているのが，V-Text APIです．このAPIの一部の機能は，Super DriversやDISPS3などでもサポートされています．

DOS/V Extensionの画面表示API（INT 10h）には以下のものがあります．

高密度テキストモードへの切り替え ------------------------

BL = 0

VIDEO　1118h 高密度テキストモードへの切り替え（モード03h，73hのみ）

この機能は，標準のテキストモード（80桁×25行）から，現在設定されている高密度モード（80桁×26行以上）へ切り替えるために使用します．ここでいう高密度モードとは，ビデオモードおよび画面幅が80桁のままで，行数だけが26行以上に拡張される状態のことをいいます．

また，現在の高密度モードの設定は，DOS/Vテキスト密度の切り替え（AH=12h, BL=39h,INT 10h）によって設定されます．

この機能は，ビデオモードの変更直後に実行してください．単純にビデオモードの設定

(AH=00h,INT 10h) を行えば，標準のテキストモードに戻ります．

この機能はSuper Driversではサポートされていません．

DOS/V拡張モードテーブルの取得

VIDEO　1131h DOS/V拡張モードテーブルの取得

[戻り値]

ES:BP　拡張モードテーブルへのポインタ

CX　テーブル項目数

各項目は16バイトで構成されていて，以下の形式になっています．

0バイト目	ビデオモード番号
1バイト目	ビデオモード情報（以下を参照）
2バイト目	画面の横幅（桁数）
3バイト目	画面の高さ（行数）
4バイト目	文字ボックスの幅（ビット数）
5バイト目	文字ボックスの高さ（ビット数）
6バイト目	半角フォントの幅（ビット数）
7バイト目	フォントの高さ（ビット数）
8～16バイト目	システム予約

このテーブルには，DOS/V拡張ビデオモード（高品位および高密度テキストモード）だけが含まれています．

ビデオモード番号が03h,73hの場合は，横幅は80桁固定で，行数は25行以上となります．また，70h，73hの場合は行数，桁数とも可変です．

ビデオモード情報は，以下の形式になっています．

ビット7	もし，1ならそのビデオモードは使用できません．
ビット6-2	必ず，0です．
ビット1-0	システム使用として予約されています．

文字ボックスの大きさとフォントの大きさに関しては，図2.11を参照してください．

この機能はSuper Driversでも使用できます．


```
****************
*
*
*
*
**************
*
*
*
*
*
*
*
```

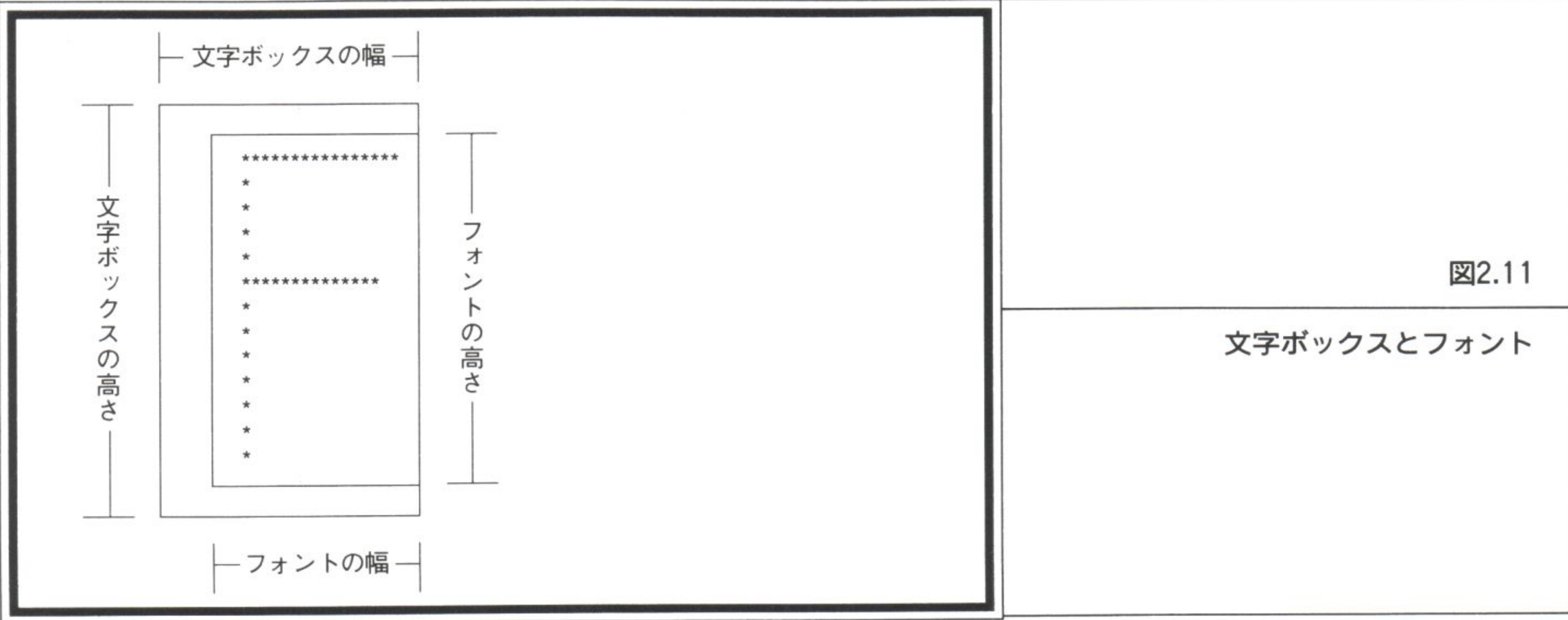



図2.11

文字ボックスとフォント

DOS/Vフォント品位の切り替え

BL =　　38h　DOS/Vフォント品位の切り替え（モード03h,73hのみ）
AL =　　拡張モードテーブルの項目番号（0,1...）
　　基本テキストモード（80桁×25行）に戻す場合は，0FFh
BH =　　ALが0FFhの場合は，ビデオモード番号
VIDEO　12h
[戻り値]
AL = 12h　この機能はサポートされている
　12h以外　この機能はサポートされていない

この機能は，内部的に設定されるだけで，画面が実際に切り替わるのは，次のビデオモード設定（モード03h，73hのみ）の場合です．

モードテーブルの項目番号は，DOS/V拡張モードテーブルの取得（AX=1131h,INT 10h）で得られたテーブルの項目番号です．ただし，必ず80桁×25行のモード03h，07hだけに限ります．もし項目番号に0FFhを指定した場合には，BHにビデオモード（03h,73h）を設定してください．このときはデフォルトの文字品位になります．

この機能はSuper Driversでは使用できません．

DOS/Vテキスト密度の切り替え

BL = 39h　DOS/Vテキスト密度の切り替え
AL =　　拡張モードテーブルの項目番号（0,1...）
　　0FFhの場合は，このモード設定を消去する
BH =　　ALが0FFhの場合は，ビデオモード番号
VIDEO　12h
[戻り値]
AL = 12h　この機能はサポートされている
　12h以外　この機能はサポートされていない

この機能は，内部的に設定されるだけで，画面が実際に切り替わるのは，次のビデオモード設定（モード70h，71hのみ）の場合と，高密度テキストモードへの切り替え（AX=1118h,INT 10h）の場合です．また，別のパラメータで呼び出されるまでこの設定値は有効です．

モードテーブルの項目番号は，DOS/V拡張モードテーブルの取得（AX=1131h,INT 10h）で得られたテーブルの項目番号です．もし項目番号に0FFhを指定した場合には，BHに設定されたビデオモードのテキスト密度設定は消去されます．

この機能は，各ビデオモード（03h,70h,71h,73h）間で独立に動作します．

この機能はSuper Driversでも使用できます．

DOS/Vモード設定の取得

BL = 3AH　DOS/Vモード設定の取得
AL =　　ビデオモード番号（03h,70h,71h,73h）
VIDEO　12h

[戻り値]

AL = 12h　　この機能はサポートされている

CH = 　　フォント品位設定（モード03h，73hの場合）

　　モードテーブル項目番号（0よりの相対）または0FFhが戻る

CL = 　　テキスト密度設定

　　モードテーブル項目番号（0よりの相対）または0FFhが戻る

フォント品位設定はALに設定されたビデオモードが03h,73hの場合だけ戻ります．それ以外の場合には0FFhが戻ります．

もし，テキスト密度設定に0FFhが戻った場合には，ALで設定されたビデオモードには可変密度テキストモードが存在しないことを意味しています．

この機能はSuper Driversでも使用できます．

非公開機能

BH = 01h　　0000:010Ch（INT 43h）のエントリアドレスを返す

VIDEO　1130h フォントテーブル情報の取得

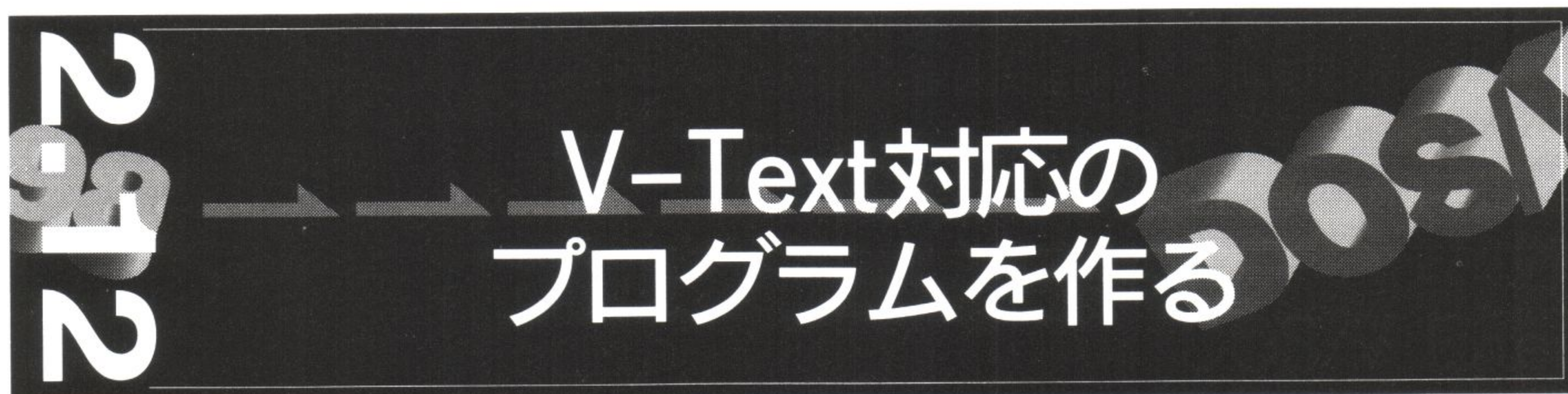
[戻り値]

CX　　文字フォントの高さ

DL　　スクロール可能行数ー1

ES:BP　　BHで指定されたアドレス

この機能は，本来はフォントテーブル情報を取得するためのものですが，同時にフォントの高さや，スクロール可能行数ー1の値を取得することもできます．同じ値はBIOSワークエリアからも取得できます．戻ってきたDLの値に，＋1とし，かな漢字変換などのために予約している行数（AX=1D02h,BX=0,INT 10hで取得できます）を足しこんだものが，実際の画面の行数です．

2.12 V-Text対応のプログラムを作る

では，本書で作成したパーツを利用して，ファイルダンプ表示のプログラムを作成してみましょう．基本的な仕様は以下のように決定しました．

◎ダンプするファイルはすべてメモリ上に読み込む（高速スクロールおよび将来の拡張のため）．

◎画面の行数に合わせて，ダンプ行数を決定する．

◎画面の幅が100桁以上ある場合は，右側にガイダンスを表示する．

このため，まずC部分の関数の構成を以下のようにします．また，基本的にこれまでに作成したアセンブラ関数を使用します．

main()	初期設定・終了処理・キー入力などを行う
ReadFileMemory()	ファイルをメモリに読み込む
DrawRuledLines()	罫線やタイトルを表示する
cls()	画面を消去する
DisplayAddr()	画面の1行目のアドレスを表示する
DumpFile()	1画面分のダンプを表示する
DumpOneLine()	DumpFileのサブルーチンとして1行分のダンプを表示する

今回のプログラムでは，エミュレートCGAテキストモードを使用して，大半の処理を仮想VRAMへの直接アクセスで行っています．スクロールのところで解説しましたが，画面まわりでもっとも気になるのがスクロールの処理方式で何を使用するかです．今回は，本来ビデオBIOSがもっているスクロール機能は一切使用していません．

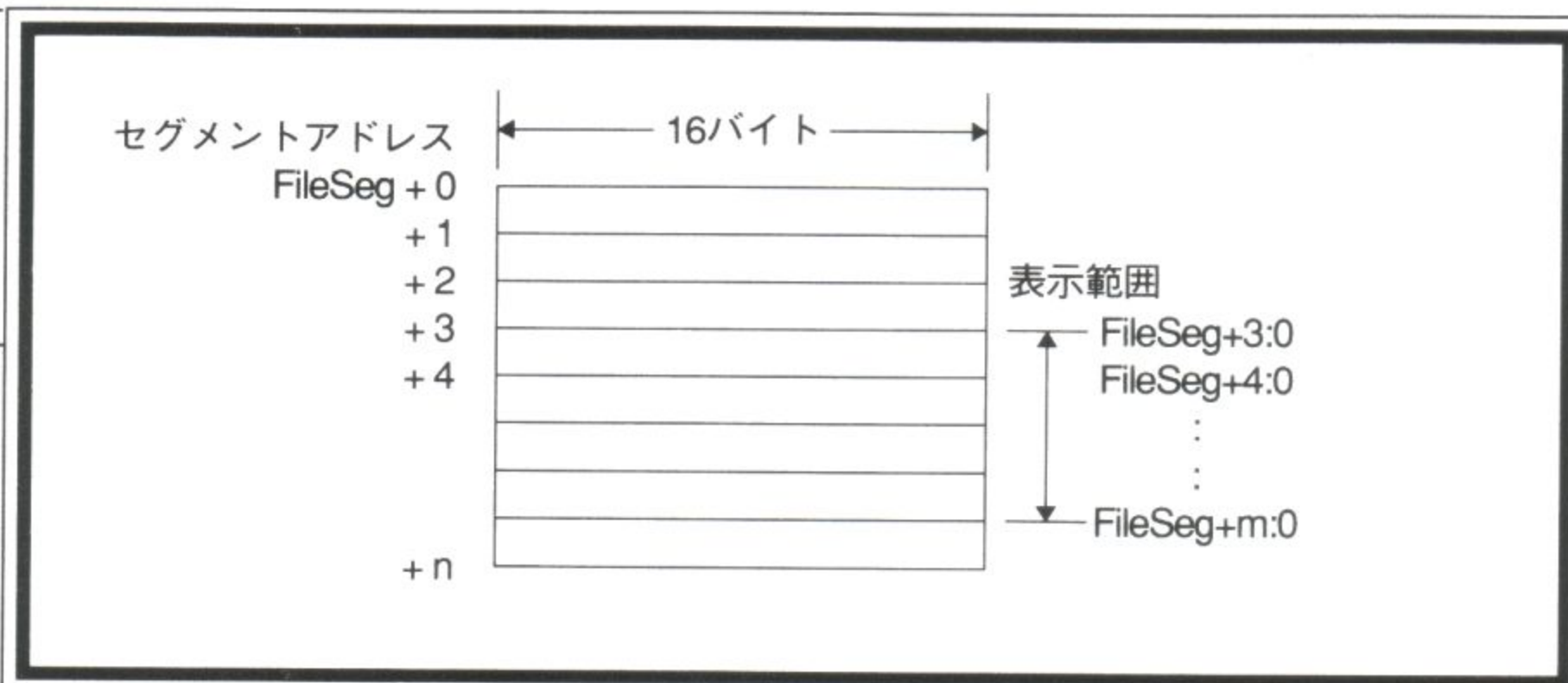
また，プログラムを読むうえで注意しなければいけない点としては，以下のものがあります．

ひとつは，64KB以上のファイルも処理可能にするため，ファイルバッファの構造を工夫しています．本来はhugeモデルを使用すればよいのですが，今回はファイルバッファの表示先頭セグメントアドレスを変更するようにして，64KBを越すファイルバッファでも同じようにアクセスさせています．すなわち，ダンププログラムなので基本的には16バイトで1行を表示します．これは，1行スクロールさせると，表示開始セグメントアドレスを+1させて表示すれば，16バイト分スクロールしたのと同等になることを利用しています．このため，ファイルバッファの先頭セグメントアドレスをFileSeg変数に，現在表示している先頭行の先頭からの差をTop変数に割り当てて，この変数+画面上での表示行数目から，その行を含むセグメントアドレスを計算して，1行ダンプ表示用の関数に渡すことにより，16バイトの表示を行っています．（DumpOneLine関数）

また，ファイルの読み込みもファイル全体を読み込めるだけのメモリを確保し，そこに32KB単位で読み込むようにしています．（ReadFileMemory関数）

ダンプ部分に関しては，アドレス（6バイト），16進数ダンプ，文字ダンプの3部分に分かれますが，通常問題になるのは漢字の泣き分かれです．すなわち，前の行の16バイト目

図2.12

ファイルバッファの構造

に漢字がある場合，次の行の1バイト目と泣き分かれになり，文字が化けてしまう場合があります．また，オブジェクトプログラムなどのダンプは漢字と同じ文字コードがあり，その部分も化けてしまう場合があります．

前者に関しては，KanjiFlag変数を使用して，前の行の最終桁が漢字1バイト目で，次の行の先頭桁が漢字2バイト目の場合は，前の行の16・17桁目に漢字を表示してしまい，次の行の1桁目には何も表示しないようにしています．スクロールアップ時には，仮想VRAM上のデータをスクロールさせていますので，先頭行の先頭桁もほぼ正確に表示されます．スクロールダウン時にはダンプ部分の全体を表示していますので，その画面内では先頭行の先頭桁を除いては，正確に表示されるはずです．

後者のオブジェクトなどのダンプの場合には，漢字1バイト目，漢字2バイト目をチェックして，漢字1バイト目の次に漢字2バイト目の範囲内の文字がこなかった場合には，漢字1バイト目に対応する文字ダンプ部分に“*”を表示し，再度漢字1バイト目かどうかをチェックするようにして，なるべく文字化けが生じないようにしています．厳密には範囲のチェックだけでは，漢字かどうかは判断できないのですが，実用上はあまり問題にならないと思います．（DumeOneLine関数）

リスト2.18　　**ファイルダンプの中心となるC部分SDP.C**

```
#include   <stdio.h>
#include   <string.h>
#include   <memory.h>
#include   <stdlib.h>
#include   <dos.h>

#define    LINE_MAIN
#include   "RomBios.h"

#define KEISEN      (ATRY|HIGHBRIGHT)                         // 罫線用の画面属性
#define DUMPT       (B_ATRB|ATRC|HIGHBRIGHT)                  // タイトル部分の画面属性
#define FNAMET      (B_ATRY|ATRB|B_HIGHBRIGHT)                // ファイル名タイトルの画面属性
#define FNAME       (HIGHBRIGHT|ATRW)                         // ファイル名の画面属性
#define READSIZE    32768L                                    // 一回に読み込むバイト数
#define READPARA    (int)(32768L/16)                          // 一回に読み込むパラグラフ数

static     VIDEO_INF  VideoInf ;                              // 初期ビデオモード格納

struct {                                                      // 罫線表示位置
           int        vc ;
           int        vpos[5] ;
} vlm ;

int                   StX ,                                   // 画面開始桁
                      VtFlag ;                                // V-Text フラグ
uchar      *vt[] = {  " Addr." ,
                                          "00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F" ,
                                          "----+----+----+-" ,
                                          "INFORMATION         " } ;

//        1         2         3         4         5         6         7         8         9         A
// 123456789+123456789+123456789+123456789+123456789+123456789+123456789+123456789+123456789+123456789+
// |123456| 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 | 0123456789abcdef | 00h = 212,345,678 dw |

                                           /* 1234567890123456789012 */
uchar      *guide[] = {          "ESC       終了" ,            // V-Textときに表示するガイダンス
                                                      "↑         16バイト上へ" ,
                                                      "↓         16バイト下へ" ,
                                                      "Shift+↑↓高速",
                                                      "Home      先頭へ" ,
                                                      "End       最後へ" ,
                                                      "PageUp    前画面へ" ,
                                                      "PageDown  次画面へ" } ;

uint       FileSeg ,                                          // ファイルバッファの先頭セグメント
           LastCount ,                                        // ファイルバッファのパラグラフ数
           RemainBytes ;                                      // 最後のパラグラフのバイト数
int        DisplayLine ,                                      // ダンプ部分の画面行数
           KanjiFlag ;                                        // 漢字1文字目かのチェックフラグ
long       FileSize ,                                         // ダンプするファイルの大きさ
           Top ,                                              // 表示する先頭のパラグラフ番号
           OldTop = -1 ,                                      // 前回表示したパラグラフ番号
           Bottom ;                                           // 最後に表示する先頭のパラグラフ番号
int        Base ;                                             // ファイルのアドレスを表示する桁数

void       ReadFileMemory(char *FileName) ;                   // ファイル入力関数
void       DrawRuledLines(void) ;                             // 罫線などを表示する関数
void       cls(void) ;                                        // 画面消去関数
void       DisplayAddr(void) ;                                // 現在のアドレスを表示する関数
```

```
void      DumpFile(void) ;                                    // 1画面分のダンプ表示関数
void      DumpOneLine(char *, long, int) ;                    // 1行分のダンプ表示関数


/**************************************************************************/
/*                                                                        */
/*                  Main Procedure Part Start                             */
/*                                                                        */
/**************************************************************************/
main(int  argc,uchar  **argv)
{

        int       ScrollCnt;                                  // スクロール行数
        uint      c ;                                         // キー入力文字
        uchar     TotalSize[10  ;                             // ファイルサイズ表示用
        int       i ;

        if ( argc < 2 ) {                                     // パラメータ数のチェック
                  printf("¥7¥nUSAGE>>sdp filename¥n") ;
                  return 1 ;
        }

        ReadFileMemory(argv[1]) ;                             // メモリ上にファイル全体を読み込む

        SetVramSegment() ;                                    // ビデオ初期化
        GetVideoInfo(&VideoInf) ;                             // ビデオ情報取得
        DisplayLine = VideoInf.Lines - 5 ;                    // 表示行数
        SaveCursor() ;                                        // カーソル位置・形状の保管
        SetCursorType(0x20,0) ;                               // カーソルを非表示に

        cls() ;                                               // 画面消去

        VtFlag = VideoInf.Column >= 100 ? 1 : 0 ;             // V-Text チェック
        StX = ( ( VideoInf.Column - ( VtFlag ? 100 : 77 ) ) >> 1 ) + 1 ; // 開始桁

        DrawRuledLines() ;                                    // 画面罫線の出力

        PutStringAbs(StX+1 ,1," File Name: ",FNAMET) ;        // ファイル名の表示
        PutStringAbs(StX+14,1,argv[1      ,FNAME ) ;

        Base = VtFlag ? 58 : 45 ;                             // ファイルアドレス表示の桁数
        PutStringAbs(StX+Base   ,1," Current Adr.: ",FNAMET) ; // アドレスタイトル
        sprintf(TotalSize, "(%06lXH)",FileSize);              // ファイルサイズの変換
        PutStringAbs(StX+Base+23, 1, TotalSize, FNAME) ;      // ファイルサイズの表示

        Top = 0;                                              // 先頭位置
        if ( ( Bottom = LastCount - DisplayLine + 1 ) < 0 )   // 最終表示位置
                  Bottom = 0 ;
        do {
                  if ( Top < 0 )          Top = 0 ;           // Top の範囲チェック
                  if ( Top > Bottom )     Top = Bottom ;

                  if ( Top != OldTop ) {                      // ファイルの表示位置が変わった場合
                            DumpFile();                       // ファイルダンプの表示
                            OldTop = Top ;
                  }
                  KeyBufClr();                                // キーボードバッファのクリアー
                  c = GetKey(NOCHECK) ;                       // キーコードの取得
                  switch (c) {
                            case      0x8050:                 // ↓キー
                                      ScrollCnt = 1 ;
                                      do {
                                                if (GetShiftStatus() & 0x03)       // シフトキーが押されているとき
                                                          ScrollCnt *= 2;          // はスクロール量を2倍にする
                                                Top += ScrollCnt ;
                                                if ( Top > Bottom ) Top = Bottom;  // Top の範囲チェック
                                                if ( Top != OldTop ) {
                                                          ScrollCnt = (int)(Top - OldTop);  // 実際のスクロール量
                                                          DisplayAddr();                    // 現在のアドレス表示
                                                          ScrollUp(StX+ 1,4,StX+75,3+DisplayLine,ScrollCnt);        // スクロール
                                                          for( i = 0 ; i < ScrollCnt ; i++ )
                                                                    DumpOneLine( (void far *)((long)(FileSeg+Top+DisplayLine-1-i) << 16),
                                                                                        Top+DisplayLine-1-i, DisplayLine+3-i);
                                                          ScreenRewrite(4,3+DisplayLine);             // 画面の再描画
                                                          OldTop = Top;
                                                }
                                                if ( (c = GetKey(0x8050)) != 0 ) {                    // 続けて↓キーが押されて
                                                          ScrollCnt = 2 ;                             // いるかのチェック
                                                          KeyBufClr();                                // 押されているなら2行
                                                }                                                     // スクロールさせる
                                      } while ( c != 0 ) ;
                                      break;
                            case      0x8048:                                                         // ↑キー
                                      ScrollCnt = 1 ;
                                      do {
                                                if (GetShiftStatus() & 0x03)                          // シフトキーが押されているとき
                                                          ScrollCnt *= 2;                             // はスクロール量を2倍にする
                                                if (( Top -= ScrollCnt ) < 0 )      Top = 0 ;
                                                if ( Top != OldTop ) {
                                                          ScrollCnt = (int)(OldTop - Top);            // 実際のスクロール量
                                                          DumpFile();                                 // ダンプの表示
                                                          OldTop = Top ;
                                                }
                                                if ( (c = GetKey(0x8048)) != 0 ) {                    // 続けて↓キーが押されて
                                                          ScrollCnt = 2 ;                             // いるかのチェック
                                                          KeyBufClr();                                // 押されているなら2行
                                                }                                                     // スクロールさせる
                                      } while ( c != 0 ) ;
                                      break;
                            case      0x8047:                                                         // Homeキー
                                      Top = 0 ;                                                       // 先頭に移動する
                                      break;
                            case      0x804F:                                                         // Endキー
```

```
                                        Top = Bottom ;                              // 最終表示位置に移動する
                                        break;
                        case    0x8049:                                             // Page Upキー
                                        Top -= (DisplayLine) ;                      // 1画面分前に移動する
                                        break;
                        case    0x8051:                                             // Page Downキー
                                        Top += (DisplayLine) ;                      // 1画面分後ろに移動する
                                        break;
                }
        } while (c != 0x1b) ;                                                       // Escキーが押されたら終了する

        cls() ;                                                                     // 画面消去

        RestoreCursor();                                                            // カーソル位置・形状の復元
        SetCursorPos(1,1);                                                          // カーソル位置の設定

        return 0 ;
}

/****************************************************************************/
/*                                                                          */
/*                      ファイルをメモリに読み込む                          */
/*                                                                          */
/****************************************************************************/
void    ReadFileMemory(char *FileName)
{

        FILE    *fp ;                                           // ファイル・ハンドル
        long    RemainSize ,                                    // まだ読み込んでいないバイト数
                        l ;
        uint    ReqParagraph ,                                  // 読み込みに必要なパラグラフ数
                        i ;
        void    *Ptr ;                                          // ファイル入力ポインタ

        if ( !( fp = fopen(FileName,"rb") ) ) {                 // ファイルがオープンできないとき
                printf("¥7¥nCan not open file <<%s>> .....¥n",FileName) ;
                exit(-1) ;
        }

        fseek(fp, 0L, SEEK_END) ;                               // ファイルの終端に移動
        FileSize = ftell(fp) ;                                  // ファイルサイズの取得
        fseek(fp, 0L, SEEK_SET) ;                               // ファイルの先頭に移動

        ReqParagraph = (uint)((FileSize + 15L) >> 4) ;          // セグメントサイズを求める
        LastCount = ReqParagraph - 1 ;                          // 最後のパラグラフ番号
        RemainBytes = (int)(( FileSize - 1 ) % 16 + 1);         // 最後のバイト数
        _dos_allocmem(ReqParagraph, &FileSeg) ;                 // 入力バッファの確保
        if ( FileSeg  == 0 ) {                                  // バッファが確保できない
                printf("¥7¥nNot enough memory!¥n") ;
                exit(-1) ;
        }

        RemainSize = FileSize ;                                 // 残りのファイルサイズ
        for ( i = 0, l = 0 ; l < FileSize ; i += READPARA, l += READSIZE ) {
                Ptr = (void far *)((long)(FileSeg + i ) << 16) ;        // far アドレスへの変換
                fread(Ptr, 1, (size_t)(RemainSize > READSIZE ? READSIZE : RemainSize), fp) ;
                                                                // ファイルの読み込み
                RemainSize -= READSIZE ;                        // 残りのファイルサイズ
        }
        fclose(fp) ;                                            // ファイルのクローズ

}

/****************************************************************************/
/*                                                                          */
/*                          罫線などの表示                                  */
/*                                                                          */
/****************************************************************************/
void    DrawRuledLines(void)
{

        int             i , j ;

        PutLoopChar(StX   ,2,DisplayRuledLine[TOP_LEFT],KEISEN,1) ;
        PutLoopChar(StX+ 1,2,DisplayRuledLine[TOP_HOLIZ],KEISEN,6) ;
        PutLoopChar(StX+ 7,2,DisplayRuledLine[CENTER_TOP],KEISEN,1) ;
        PutLoopChar(StX+ 8,2,DisplayRuledLine[TOP_HOLIZ],KEISEN,49) ;
        PutLoopChar(StX+57,2,DisplayRuledLine[CENTER_TOP],KEISEN,1) ;
        PutLoopChar(StX+58,2,DisplayRuledLine[TOP_HOLIZ],KEISEN,18) ;

        vlm.vpos[0] = StX ;
        vlm.vpos[1] = StX +  7 ;
        vlm.vpos[2] = StX + 57 ;
        vlm.vpos[3] = StX + 76 ;
        vlm.vpos[4] = StX + 99 ;

        if ( VtFlag ) {
                PutLoopChar(StX+76,2,DisplayRuledLine[CENTER_TOP],KEISEN,1) ;
                PutLoopChar(StX+77,2,DisplayRuledLine[TOP_HOLIZ],KEISEN,22) ;
                PutLoopChar(StX+99,2,DisplayRuledLine[TOP_RIGHT],KEISEN,1) ;
                vlm.vc = 4 ;
        }
        else    {
                PutLoopChar(StX+76,2,DisplayRuledLine[TOP_RIGHT],KEISEN,1) ;
                vlm.vc = 3 ;
        }

        for ( i = 3 ; i < VideoInf.Lines - 1 ; i ++ ) {         // Lines - 3 まで
                for ( j = 0 ; j < vlm.vc ; j ++ )
                        PutLoopChar(vlm.vpos[j],i,DisplayRuledLine[LEFT_VERT],KEISEN,1) ;
                PutLoopChar(vlm.vpos[vlm.vc],i,DisplayRuledLine[RIGHT_VERT],KEISEN,1) ;
        }

        PutLoopChar(StX   ,VideoInf.Lines-1,DisplayRuledLine[BOTTOM_LEFT],KEISEN,1) ;
```

```
		PutLoopChar(StX+ 1,VideoInf.Lines-1,DisplayRuledLine[BOTTOM_HOLIZ],KEISEN,6) ;
		PutLoopChar(StX+ 7,VideoInf.Lines-1,DisplayRuledLine[CENTER_BOTTOM],KEISEN,1) ;
		PutLoopChar(StX+ 8,VideoInf.Lines-1,DisplayRuledLine[BOTTOM_HOLIZ],KEISEN,49) ;
		PutLoopChar(StX+57,VideoInf.Lines-1,DisplayRuledLine[CENTER_BOTTOM],KEISEN,1) ;
		PutLoopChar(StX+58,VideoInf.Lines-1,DisplayRuledLine[BOTTOM_HOLIZ],KEISEN,18) ;

		if ( VtFlag ) {
			PutLoopChar(StX+76,VideoInf.Lines-1,DisplayRuledLine[CENTER_BOTTOM],KEISEN,1) ; ;
			PutLoopChar(StX+77,VideoInf.Lines-1,DisplayRuledLine[BOTTOM_HOLIZ],KEISEN,22) ;
			PutLoopChar(StX+99,VideoInf.Lines-1,DisplayRuledLine[BOTTOM_RIGHT],KEISEN,1) ;
		}
		else	PutLoopChar(StX+76,VideoInf.Lines-1,DisplayRuledLine[BOTTOM_RIGHT],KEISEN,1) ;

		PutStringAbs(vlm.vpos[0]+1,3,vt[0],DUMPT) ;			// タイトルの表示
		for ( i = 1 ; i < vlm.vc ; i++ )
			PutStringAbs(vlm.vpos[i]+2,3,vt[i],DUMPT) ;

		if ( VtFlag ) {							// ガイダンスの表示
			for ( i = 0 ; i < 8 ; i++ )
				PutStringAbs(StX+77,VideoInf.Lines-10+i,guide[i],FNAME) ;
		}

}

/****************************************************************************/
/*									*/
/*			画面の消去						*/
/*									*/
/****************************************************************************/
void	cls(void)
{
		PutLoopChar(1,1,' ',ATRW,VideoInf.Column * VideoInf.Lines) ;
}


/****************************************************************************/
/*									*/
/*			現在アドレスの表示					*/
/*									*/
/****************************************************************************/
void	DisplayAddr(void)
{
		uchar	CurAddr[8];

		sprintf(CurAddr, "%05lX0H",Top);		// 現在のアドレスを変換
		PutStringAbs(StX+Base+16, 1, CurAddr, FNAME) ; // 現在のアドレスの表示
}


/****************************************************************************/
/*									*/
/*			ファイルダンプの表示					*/
/*									*/
/****************************************************************************/
void	DumpFile()
{
		int	i;
		char	*ptr ;

		DisplayAddr();						// 現在のアドレス表示

		ptr = (void far *)((long)(FileSeg+Top) << 16);		// バッファへのポインタ

		KanjiFlag = 0 ;						// 漢字1文字目フラグ
		for ( i = 0 ; i < DisplayLine ; i++ ) {
			DumpOneLine(ptr , Top+i , i+4);			// 1行表示
			ptr += 16 ;
		}
		ScreenRewrite(4, 3+DisplayLine);			// 仮想VRAM再表示
}


/****************************************************************************/
/*									*/
/*			ダンプ1行の表示						*/
/*									*/
/****************************************************************************/
void	DumpOneLine(char *ptr, long adr, int line)
{
		int	jend ,						// この行のバイト数
			j ;
		uchar	c ,
			c1 ,
			Addr[7 ,					// アドレス部分
			Hex[49 ,					// 16進数ダンプ表示部分
			Ascii[18 ;					// 文字表示部分

		memset(Hex,0x20,sizeof(Hex)) ;				// バッファの消去
		memset(Ascii,0x20,sizeof(Ascii)) ;			// バッファの消去

		sprintf(Addr, "%05lX0",(long)adr);			// アドレスの変換

		jend = ( LastCount == (uint)adr ) ? RemainBytes : 16 ;	// この行のバイト数
		for ( j = 0 ; j < jend ; j++ ) {
			sprintf(&Hex[j*3], " %02X", (c = *ptr++)) ;	// 16進数への変換
			if ( KanjiFlag == 0 ) {				// 漢字2文字目ではないとき
				if ( c < 0x20 ) {			// 0x20以下の文字
					Ascii[j = c ;
				}
				else {					// 通常の文字
					Ascii[j = c ;
					if (( c >= 0x81 && c <= 0x9f ) ||	// 漢字1文字目の判断
						( c >= 0xe0 && c <= 0xfc )) {
						c1 = *ptr ;
```

```
                                                        if ( cl >= 0x40 && cl <= 0xfe && cl != 0x7f ) {// 漢字2文字目の判断
                                                                Ascii[j+1] = cl ;
                                                                KanjiFlag = 1 ;
                                                        }
                                                        else                    // 漢字2文字目ではないとき
                                                                Ascii[j] = '*' ;
                                                }
                                        }
                                }
                                else                                            // 漢字2文字目はすでに表示済み
                                        KanjiFlag = 0 ;
                        }
                        if ( j < 16 ) Hex[j*3] = ' ' ;                          // 後続のブランクと結合する

                        PutStringAbs2(StX+ 1, line, Addr , ATRW,  6) ;           // アドレスの仮想VRAMへのセット
                        PutStringAbs2(StX+ 8, line, Hex  , ATRW, 48) ;           // 16進数の仮想VRAMへのセット
                        PutStringAbs2(StX+59, line, Ascii, ATRW, 17) ; ..        // 文字の仮想VRAMへのセット
}
```

2.13 グラフィック処理

ビデオBIOSにも，グラフィック画面に対するドットの書き込み（AH=0Ch）とドットの読み取り機能（AH=0Dh）がありますが，この機能だけでは当然グラフィック操作は，表示が極端に遅くなり，実用に絶えません．そこで，ここでは簡単にVGAグラフィックの基礎とVESA機能に関して解説します．

◎PC/ATのグラフィック画面には，ビデオモードに応じて，複雑な表示能力をもっています．

◎グラフィックVRAMは色数により，複数のプレーンに別れており，アクセスするためには，必ず，VGAグラフィックコントローラを設定しなければなりません．

◎VGA以上の規格に関しては，VESAでまとめられつつあります．

基本となるVGAは，MDA，CGA，EGAという3つのビデオシステムがもっていたビデオモードをサポートしていますし，それに加えて，グラフィックに限れば，640×480ドット16色モードや320×200ドット256色モードをもっています．

ここでは，話しをわかりやすくするために，640×480ドット16色モードに関して解説します．VGAでは，PC-9801と異なり，直接CPUからグラフィックVRAMをアクセスすることはできません．必ず，VGAグラフィックコントローラを設定してから，グラフィックVRAMを操作するようにします．すなわち，将来的な拡張およびコンベンショナルメモリを圧迫しないようにするため，VGAのグラフィックVRAMは実際には色ごとにカラープレーンとして別れており，かつ，すべてのプレーンは同一のアドレスを占めています．そのため，CPUがグラフィックVRAMを直接アクセスすることはそのままでは意味がありません．

VGAグラフィックコントローラ内部には，9個のレジスタ（8ビット長，表2.10を参照）と4個のラッチ（8ビット長）があります．カラープレーンにアクセスするためには，まず，グラフィックコントローラ内部のレジスタに必要な値を設定して，CPUメモリ空間にあるグラフィックVRAMにアクセスします．そのアクセスした8ビットのデータとアドレ

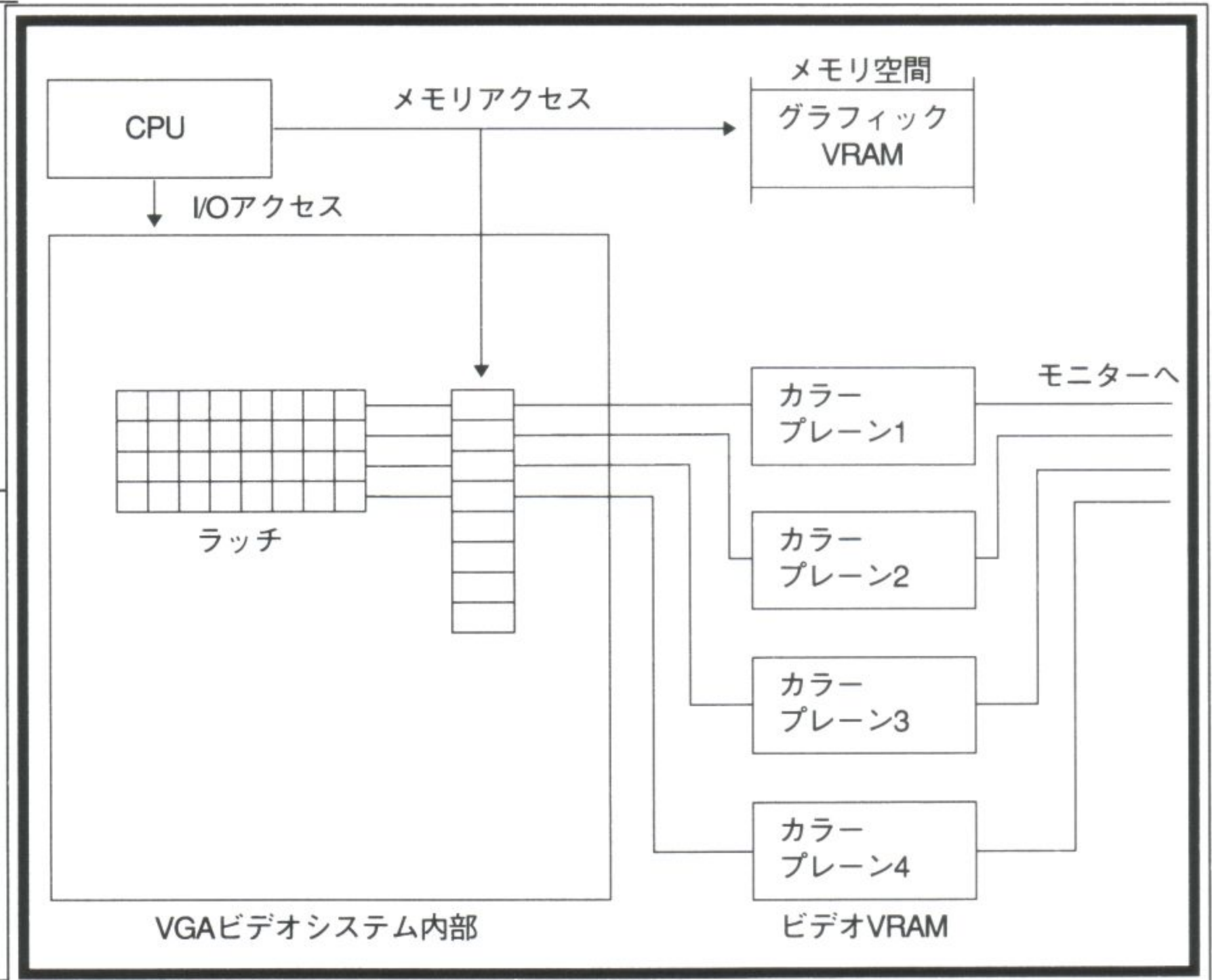
図2.13
VGAグラフィックのしくみ

スを使用して，レジスタの内容にしたがって，各種演算を行いながら，ラッチに各カラープレーンの値を読み込んだり，ラッチの値と組み合せてカラープレーンに書き込んだりします．

図2.13にVGAグラフィックのしくみを示しておきます．

グラフィックコントローラには，アドレスレジスタとデータレジスタの2つがI/Oアドレス空間に割り当てられていて，アドレスレジスタに内部レジスタ番号をセットすると，そのレジスタはデータレジスタのI/Oアドレスを使用してアクセスできるようになっています．

実際のコーディング関連やグラフィック操作に関して詳細に知りたい場合には，翔泳社から発行されている『PC&PS/2ビデオシステムプログラマズガイド』が参考になると思います．

表2.10 VGAグラフィック制御レジスタ

レジスタ番号	名称		デフォルト値
0	Set/Reset		0
	ビット7-4	予約済み	
	ビット3	プレーン3	
	ビット2	プレーン2	
	ビット1	プレーン1	
	ビット0	プレーン0	
1	EnableSet/Reset		0
	ビット7-4	予約済み	
	ビット3	プレーン3	
	ビット2	プレーン2	
	ビット1	プレーン1	
	ビット0	プレーン0	
2	ColorCompare		0
	ビット7-4	予約済み	
	ビット3	プレーン3	
	ビット2	プレーン2	
	ビット1	プレーン1	
	ビット0	プレーン0	

3	Data Rotate/Function Select	0
	ビット7-5　予約済み	
	ビット4-3　論理演算指定	
	00　演算対象と置き換え	
	01　ラッチと演算対象をAND	
	10　ラッチと演算対象をOR	
	11　ラッチと演算対象をXOR	
	ビット2-0　ローテイトビット数	
4	ReadMapSelect	0
	ビット7-2　予約済み	
	ビット1-0　読みだしプレーン指定	
5	Mode	ビデオモード0-3は常に0
	ビット7　予約済み	
	ビット6　256色カラー表示	
	ビット5　シリアルモード(CGA互換)	
	ビット4　アドレスモード(CGA互換)	
	ビット3　読み出しモード	
	ビット2　予約済み	
	ビット1-0　書き込みモード	
6	Miscellaneous	ビデオモードによる
	ビット7-4　予約済み	
	ビット3-2　ビデオメモリアドレス	
	00　A0000-BFFFF	
	01　A0000-AFFFF	
	10　B0000-B7FFF	
	11　B8000-BFFFF	
	ビット1　アドレスモード(CGA互換)	
	ビット0　グラフィックスモード	
7	ColorDon'tCare	0Fh(16色モード)
	ビット7-4　予約済み01h(640×4802色モード)	
	ビット3　プレーン3	
	ビット2　プレーン2	
	ビット1　プレーン1	
	ビット0　プレーン0	
8	BitMask	FFh
	ビット7-0　書き込みビットマスク	

表2.11に，VESAでまとめられつつある，SVGA VESA BIOSの機能一覧をまとめておきます．

表2.11 VESA機能一覧

◎SVGA 情報の取得

［呼び出し方法］

ES:DI =		SVGA情報のための256バイトのバッファ
VIDEO	4F00h	

［戻り値］

AL =	4Fh	この機能はサポートされている
AH =		処理結果
	00h	正常終了
	01h	異常終了

SVGA情報のレイアウト(ES:DI)

オフセット	サイズ	記　述
00h	4 バイト	識別子('VESA')
04h	1 ワード	VESA バージョン番号
06h	ダブルワード	OEM名へのポインタ "761295520" for ATI
0Ah	4 バイト	ケーパビリティ
0Eh	ダブルワード	サポートしているVESAおよび OEMビデオモードのリストへのポインタ (リストの最後は,FFFFhで終了)
12h	238 バイト	予約済み

◎SVGA モード情報の取得

［呼び出し方法］

CX =		SVGA ビデオモード
ES:DI =		SVGAモード情報のための256バイトのバッファ
VIDEO	4F01h	

［戻り値］

AL =	4Fh	この機能はサポートされている
AH =		処理結果
	00h	正常終了
	01h	異常終了

SVGAモード情報のレイアウト(ES:DI)

オフセット	サイズ	記　述	
00h	1 ﾜｰﾄﾞ	モード属性	
		ﾋﾞｯﾄ 0	モードをサポートしている
		ﾋﾞｯﾄ 1	オプション情報が利用可能
		ﾋﾞｯﾄ 2	BIOS 出力をサポート
		ﾋﾞｯﾄ 3	1 = カラー,0 = モノクロ
		ﾋﾞｯﾄ 4	1 = グラフィック,0 = テキスト
02h	1 ﾊﾞｲﾄ	ウィンドウAの属性	
		ﾋﾞｯﾄ 0	存在している
		ﾋﾞｯﾄ 1	読み取り可能
		ﾋﾞｯﾄ 2	書き込み可能
		ﾋﾞｯﾄ3-7	予約済み
03h	1 ﾊﾞｲﾄ	ウィンドウBの属性(Aと同じ)	
04h	1 ﾜｰﾄﾞ	ウィンドウのドット数？(K)	
06h	1 ﾜｰﾄﾞ	ウィンドウサイズ(K)	
08h	1 ﾜｰﾄﾞ	ウィンドウAの開始セグメント	
0Ah	1 ﾜｰﾄﾞ	ウィンドウBの開始セグメント	
0Ch	ﾀﾞﾌﾞﾙﾜｰﾄﾞ	FARウィンドウポジショニング機能へのポインタ	
10h	1 ﾜｰﾄﾞ	走査線当たりのバイト数	

--- 以下は,VESA モード V1.0/1.1でのオプション,OEMモードで必要 ---

12h	1 ﾜｰﾄﾞ	解像度（幅）	
14h	1 ﾜｰﾄﾞ	解像度（高さ）	
16h	1 ﾊﾞｲﾄ	文字ボックスの幅	
17h	1 ﾊﾞｲﾄ	文字ボックスの高さ	
18h	1 ﾊﾞｲﾄ	メモリプレーンの数	
19h	1 ﾊﾞｲﾄ	1ピクセル当たりのビット数	
1Ah	1 ﾊﾞｲﾄ	バンクの数	
1Bh	1 ﾊﾞｲﾄ	メモリモデルタイプ	
		00h	テキスト
		01h	CGA グラフィックス
		02h	HGC グラフィックス
		03h	16色（EGA）グラフィックス
		04h	パックドピクセルグラフィックス
		05h	"sequ 256"（non-chain 4）グラフィックス
		06h	ダイレクトカラー (HiColor, 24ビットカラー)
		07h	YUV
		08h-0Fh	VESA用に予約済み
		10h-FFh	OEMメモリモデル
1Ch	1 ﾊﾞｲﾄ	バンクサイズ(KB)	
1Dh	1 ﾊﾞｲﾄ	イメージページ数	
1Eh	1 ﾊﾞｲﾄ	予約済み(0)	

--- VBE v1.2+ ---

1Fh	1 ﾊﾞｲﾄ	赤マスクサイズ
20h	1 ﾊﾞｲﾄ	赤フィールドサイズ
21h	1 ﾊﾞｲﾄ	緑マスクサイズ
22h	1 ﾊﾞｲﾄ	緑フィールドサイズ
23h	1 ﾊﾞｲﾄ	青マスクサイズ
24h	1 ﾊﾞｲﾄ	青フィールドサイズ
25h	1 ﾊﾞｲﾄ	予約済みマスクサイズ
26h	1 ﾊﾞｲﾄ	予約済みマスク位置
27h	1 ﾊﾞｲﾄ	ダイレクトカラーモード情報
28h	1 ﾊﾞｲﾄ	予約済み(0)

◎SVGA ビデオモードの設定
[呼び出し方法]
BX = ビデオモード
ビット15がオンのときは画面を消去しません.
VIDEO 4F02h

[戻り値]
AL = 4Fh この機能はサポートされている
AH = 処理結果
00h 正常終了
01h 異常終了

◎SVGA ビデオモードの取得
[呼び出し方法]
VIDEO 4F03h

[戻り値]
AL = 4Fh この機能はサポートされている
AH = 処理結果
00h 正常終了
01h 異常終了
BX = ビデオモード

◎SVGA ビデオ状況の保管・復元
[呼び出し方法]
DL = サブ機能
00h ビデオ状況バッファサイズの取得
[戻り値]:BX = 必要な64バイトブロックの数
01h ビデオ状況の保管
ES:BX = バッファ
02h ビデオ状況の復元
ES:BX = バッファ
CX = 保管・復元する状況のフラグ
ビット0: ビデオハードウェア状況
ビット1: ビデオBIOSデータ状況
ビット2: ビデオDAC状況
ビット3: SVGA状況
VIDEO 4F04h

[戻り値]
AL = 4Fh この機能はサポートされている
AH = 処理結果
00h 正常終了
01h 異常終了

◎SVGA ビデオメモリウィンドウの制御
[呼び出し方法]
BH = サブ機能
00h ビデオメモリウィンドウの設定
DX = ビデオメモリ中ウィンドウアドレス
01h ビデオメモリウィンドウの取得
[戻り値]:DX = ビデオメモリ中ウィンドウアドレス
BL = ウィンドウ番号
00h ウィンドウA
01h ウィンドウB
VIDEO 4F05h

[戻り値]
AL = 4Fh この機能はサポートされている
AH = 処理結果
00h 正常終了
01h 異常終了

◎論理走査線長の取得・設定(v1.1)
[呼び出し方法]
BL = サブ機能
00h 走査線長の設定
CX = 希望する画面ピクセル幅
01h 走査線長の取得

```
        VIDEO    4F06h

        [戻り値]
        AL =     4Fh      この機能はサポートされている
        AH =              処理結果
                            00h        正常終了
                            01h        異常終了
        BX =              走査線当たりのバイト数
        CX =              走査線当たりのピクセル数
        DX =              最大の走査線数
```

注：希望する幅が設定できない場合,つぎに大きな幅が設定されます.
　　この機能はテキストモードでも有効です.

◎表示開始位置の取得・設定（v1.1）

```
  [呼び出し方法]
        BH =     00h      予約済み
        BL =              サブ機能
                            00h        表示開始位置の設定
                                         CX = 走査線内の左端位置
                                         DX = 最初に表示する走査線
                            01h        表示開始位置の取得
        VIDEO    4F07h

  [戻り値]
        AL =     4Fh      この機能はサポートされている
        AH =              処理結果
                            00h        正常終了
                            01h        異常終了
        (BL=01hで呼び出したとき)
        BH =     00h
        CX =              走査線内の左端位置
        DX =              最初に表示する走査線
```

注：この機能はテキストモードでも有効です.

◎DACパレットコントロールの取得・設定（v1.2+）

```
  [呼び出し方法]
        BL =              機能
                            00h        DACパレット幅の設定
                                         BH = プライマリカラーに対する希望するビット数
                            01h        DACパレット幅の取得
        VIDEO    4F08h

  [戻り値]
        AL =     4Fh      この機能はサポートされている
        AH =              処理結果
                            00h        正常終了
                            01h        異常終了
        BH =              現在のプライマリカラーに対するビット数（06h = 標準VGA）
```

第3章

98

98からDOS/Vの世界へ

DOS/V

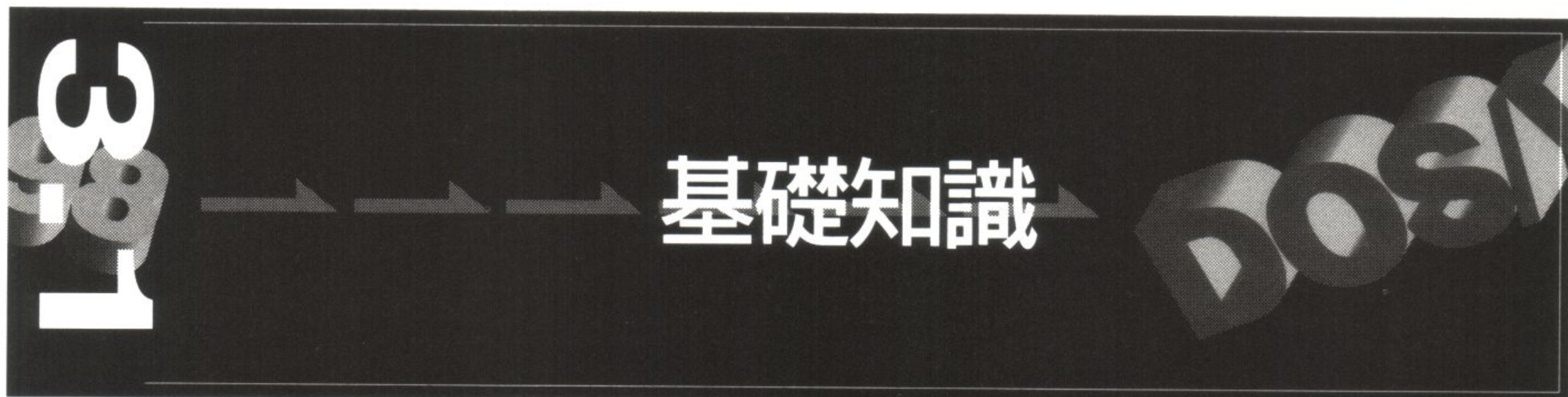
98

DOS/V

この章では，BIOSを用いたキーボード処理について解説します．BIOSキー入力では，キーボードからの文字読み込み，拡張シフトステータス状況の読み取り，キー入力センス，キータイプ（リピート）の設定，キーバッファへの書き込みなどのファンクションを解説しながら，関連するBIOSワークエリアについても解説します．

さらに，キーボードインターセプト（int 15h）を利用した走査コードの置き替えに関しても解説します．

キーボード編

3-1 基礎知識

キーのタイプには，次の3種類があります．

◎メークキー：押されたときだけ，一度だけメークコードが送られるキー

◎メークブレークキー：押されたときに一度だけメークコードが送られ，離されたときにブレークコードが送られるキー

◎タイパマティックキー：押されたときにメークコードが送られ，押し続けるとタイパマティックレートで同じメークコードがくり返し送られ，離されたときにブレークコードが送られるキー

DOS/V設定時には，[Pause]キーだけがメークキーとなり，その他のすべてのキーはメークブレーク型で，かつタイパマティックキーとなっています．

たとえば，右シフトキーが押されたとしましょう．そうすると，キーボードハードウェア割り込みが発生し，右シフトメークコードが得られます．割り込みルーチンでは，BIOSデータエリア内のシフト状況バイトのうち，右シフトビットをオンにします．つぎに“8”のキーが押されたとすると，同様に“8”に対応したスキャンコード09hが得られます．割り込みルーチンでは，BIOSデータエリア内のシフト状況を見て，スキャンコードが09hなので，スキャンコードと内部変換されたASCII文字“（”を，BIOSデータエリアのキー入力バッファに書き込みます（ただし，KEYB.COM JP組み込み時の例）．

キーボードの種類

IBM DOS J5.0/V BIOSインターフェース解説書によれば，以下の5種類のキーボードをサポートすることになっています（図3.1）．

◎IBM 5576-A01型キーボード

◎IBM 5535-S型キーボード（PS/55note，Think Padを含む）

◎IBM 5576-002型キーボード

◎IBM 5576-003型キーボード

◎IBM 5576-001型キーボード

このうち，5535-S型は5576-A01型から，5576-003型は5576-002型からテンキー部分を除いたものとなっています．したがって，Num Lock状態のときはデータキーの中央部分が数値キーパッドの代用として機能します．5576-001型は，旧来からの5550系のキーボードであり，いわゆるワープロ専用・端末専用のキーが数多くあります．したがって，IBMの汎用機やオフコンを使用している企業では数多く使用されていますが，SAA（システムア

◎IBM 5576-A01型

◎IBM 5535-S型

◎IBM 5576-002型

◎IBM 5576-003型

◎IBM 5576-001型

図3.1

IBMでサポートするキーボードのレイアウト

プリケーション体系）で規定されているキーボードとは異なりますので，将来的にはなくなるはずです．また，筆者としても，早期になくなることを期待しています．

また，PCオープンアーキテクチャー協議会（OADG）の仕様では，以下の4種類のキーボードをサポートすることになっています（図3.2）．

◎IBM 5576-A01型キーボード（OADG 106）

◎IBM US Englishキーボード（AT 101）

◎AXキーボード（AX 105）

◎J3100キーボード

IBM DOSバージョンJ5.02/Vでは，前の2つのキーボードに関しては標準で，AXキーボ

図3.2

OADGでサポートするキーボードのレイアウト（IBM 5576-A01型は図3.1を参照）

◎IBM US English

注　各キーの右上，右下の文字は，実際には核印されていません。

◎AX

◎J3100

図3.3

AXキーボード

CONFIG.SYS内での$IAS.SYSの定義方法
DEVICE=C:¥DOS¥$IAS.SYS /R=1 /K=AX

機能			キー
漢字			漢字
ローマ字 オン	←→	オフ	Ctrl + 英数カナ
ひらがな	──→	カタカナ	英数カナ
カタカナ	──→	ひらがな	Shift + 英数カナ
英数	←→	カタカナ	英数カナ
英数	←→	ひらがな	Shift + 英数カナ
半角	←→	全角	Shift + 漢字
機能なし			AXキー

図3.4

J3100キーボード

CONFIG.SYS内での$IAS.SYSの定義方法
DEVICE=C:¥DOS¥$IAS.SYS /R=1 /K=J3

機能			キー
漢字			漢字
ローマ字 オン	←→	オフ	Ctrl + 英数カナ
ひらがな	──→	カタカナ	英数カナ
カタカナ	──→	ひらがな	Shift + 英数カナ
英数	←→	カタカナ	英数カナ
英数	←→	ひらがな	Shift + 英数カナ
半角	←→	全角	Shift + 漢字
変換			スペース
無変換			Ctrl + スペース

VGA対応のJ3100ノートブック型以外のキーボードはサポートしていません．

ード（図3.3）とJ3100キーボード（図3.4）に関してはおまけのサンプルプログラムとして添付されています

このうち，DOS/Vとして標準的に使用されるのが，OADG 106キーボードですが，このキーボードは，US 101キーボードを日本語用に拡張して，「カタカナ／ひらがな」シフトキー，かな漢字変換用の「変換」，「無変換」キー，および2つのデータキーを追加しています．

キーボード割り込み処理

キーボードには専用のマイクロプロセッサが搭載され，PC本体とは双方向シリアル通信を行っています．キーがオン／オフされるごとにメークブレークコードとして，PC本体の入力ポートに送信されます．これにともない，システムボード上のキーボード制御回路が，INT09hを発生させます．このとき，I/Oポート60h（バイト）にキーコードがセットされています．

IBM PC系機種では，基本的にキーボード処理はROM BIOSが処理しますが，DOS/Vでは，2段階の拡張が行われています．

1番目は，SBCSコードジェネレータです．IBM PC系で共通に多国語対応するために，半角（SBCS）文字処理を拡張する目的で，KEYB.COMが追加されています．KEYB.COMは指定されたキーボードコードおよびコードページ（各国の半角文字の割り当てコード）に基づき，現在のシフト状況と押されたキーの組み合せにより，走査コードおよび文字コード（1バイト文字．ただし，カタカナは除く）の2バイトの組み合せ，または，ファンクションキーなどの場合には拡張コードを生成して，キー入力バッファ（BIOSワークの40h:1Ehから16ワード）にセットします．

2番目は，DBCSコードジェネレータです．DBCSコードジェネレータは日本語の入力機能を処理し，かな漢字変換が動作中はかな漢字変換結果の文字を，停止中はSBCS文字コードを自分内部にもつキー入力バッファに書き込みます．通常，キーボードBIOSで読み出

キーボード
INT 09h
キーボード割り込み
ROM BIOS /KEYB.COM
INT 15h(AH=4Fh)
キーボード
インターセプト
SBCSコードジェネレータ
INT 16h(元の)
DBCS コード生成
DBCSコードジェネレータ
($IAS.SYS / 日本語FEP)
INT 16h
アプリケーション

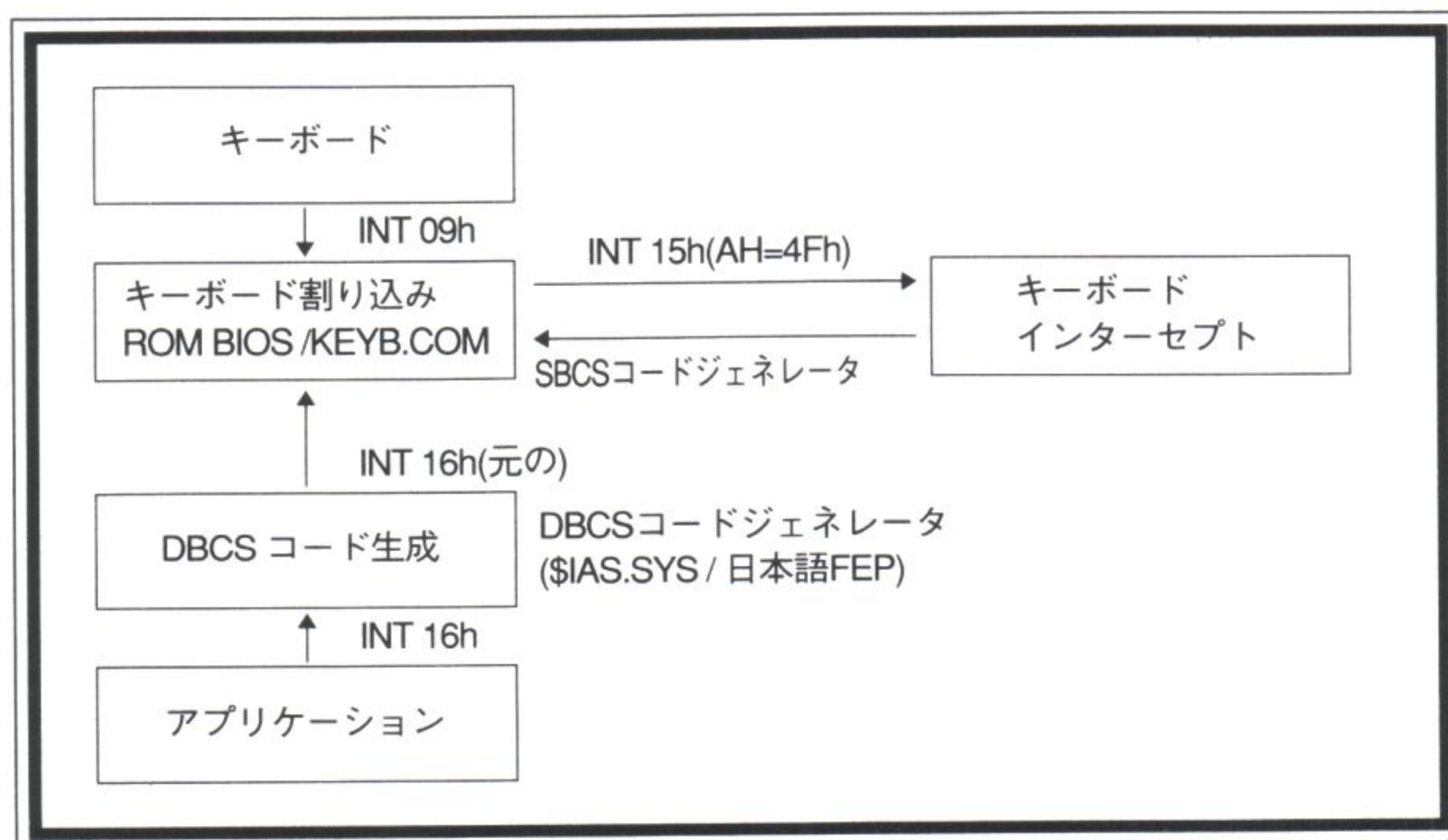



図3.5

キーボード割り込み処理

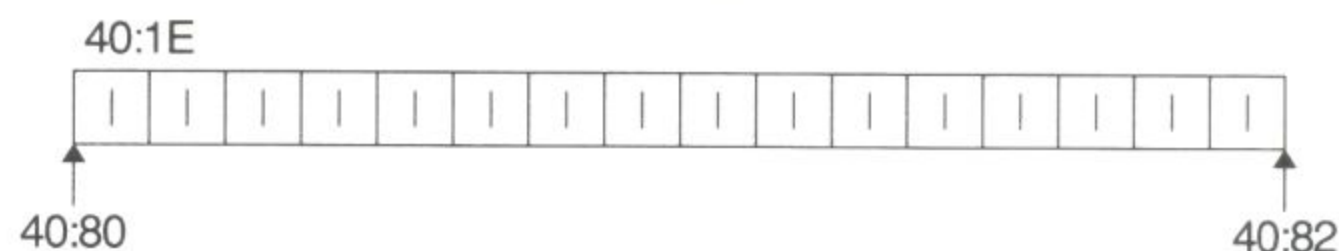
図3.6

キー入力バッファ構造

0040h:001Ah	キー入力バッファ開始ポインタ(読み取り位置)	1ワード
0040h:001Ch	キー入力バッファ終了ポインタ(書き込み位置)	1ワード
0040h:001Eh	キー入力バッファ	16ワード
0040h:0080h	キー入力バッファ先頭ポインタ	1ワード
0040h:0082h	キー入力バッファ末尾ポインタ	1ワード

40:1E

40:80

40:82

せるコードは，このDBCSコードジェネレータ内のキー入力バッファからです．また，かな漢字変換用のキー操作自身は，DBCSコードジェネレータ自身が処理してしまうため，キーボードBIOSでは通常読み取ることはできません．

したがって，PC-9801とは異なり，SBCS文字も日本語文字もキー入力バッファ（ただし，DBCS用）にセットされるため，キーボードBIOSを使用して文字入力を行う場合でも，漢字の入力が可能となっています．

キー入力バッファは図3.6のような構造になっており，2バイト×16個のリングバッファとして使用されています．そして，現在どこまで書き込まれているかを管理しているのが終了ポインタで，どこまで読み取ったのかを管理しているのが開始ポインタです．そして，終了ポインタは開始ポインタの直前まで書き込めますので，最大15文字まで記憶させることができ，これ以上入力があった場合には，ビープ音をならします．また，このビープ音自身を止めることはできません．そして，開始ポインタ＝終了ポインタ時はキー入力がないことを示しています．詳細は後述しますが，このBIOSワークのキー入力バッファでは，SBCS文字しか処理できないことに注意してください．

DOS/Vの各モードにおけるキーボードの走査コードは，走査コードセットによって決定されています．この走査コードセットには，以下のものがあります．

OADGで規定されているキーボードおよび5535-S型キーボード（PS/55note，Think Padを含む）の場合は，DOS/Vすべてのモードにおいて，走査コードセット1または2が使用されます（これは，システムユニットに搭載されているキーボードコントローラの種類に依存しています）．

5576-001型，5576-002型，5576-003型の場合には，DOS/Vの日本語モードとCHEVコマンドでUSモードにしたときは，走査コードセット81または82が使用されます（同様にキーボードコントローラに依存します）．また，SWITCHコマンドでUSモードにした場合には，走査コードセット1または2が使用されます．

しかし，実際にI/Oポート60hより読み込まれる走査コードセットはつねに1（81）になります．

これは，トランスレートモードをもつ8042をキーボードコントローラとして使用する場合には，キーボードより送られてくる走査コードセット2を8042が走査コードセット1に変換するからです．また，トランスレートモードをもたないキーボードコントローラの場合は，POST（Power On Self Test：電源投入時の自己診断テストのこと）中にキーボードにコマンドを送り，キーボードの走査コードセットを1に設定するためです．

また，5576-00x型キーボードの場合には，走査コードセット1のままでは，日本語入力に必要なキー走査コードを取得することができないため，KEYB.COMが走査コードセット81に設定しています．

また，走査コードセット1の内容を表3.2にまとめておきます．

表3.1 **走査コードセット変換**

		電源投入	POST	KEYB
キーボード	0ADG系	2	2	2
キーボードコントローラ	8042	1	1	1
キーボード	0ADG系	2	1	1
キーボードコントローラ	非8042	?	1	1
キーボード	5576-00x系	2	2	81
キーボードコントローラ	8042	1	1	81
キーボード	5576-00x系	2	1	81
キーボードコントローラ	非8042	?	1	81

表3.2 **走査コードセット1**

表の見方
キー番号は10進数，その他は16進数です．XX/YYのうち，XXが走査コードを，YYが文字コードを現します．

- -1がセットされている部分は，キーボードルーチンによって無効とされ読み込むことはできません．
- 文字コードの後ろに，*がついているものは，標準機能(AH=00h，AH=01h)では読み込めず，拡張機能(AH=10h，AH=11h)を使用した場合に読み込めます．
- 文字コードの後ろに，+がついているものは，拡張機能の場合には，文字コード部分にE0hがセットされますが，標準機能の場合には00hがセットされます．

番号	下段	上段	カナ下段	カナ上段	Alt	Ctrl	メーク	ブレーク
2	1 02/31	! 02/21	ﾇ 00/C7	-1	78/00	-1	02	82
3	2 03/32	" 03/22	ﾌ 00/CC	-1	79/00	03/00	03	83
4	3 04/33	# 04/23	ｱ 00/B1	ｧ 00/A7	7A/00	-1	04	84
5	4 05/34	$ 05/24	ｳ 00/B3	ｩ 00/A9	7B/00	-1	05	85
6	5 06/35	% 06/25	ｴ 00/B4	ｪ 00/AA	7C/00	-1	06	86
7	6 07/36	& 07/26	ｵ 00/B5	ｫ 00/AB	7D/00	07/1E	07	87
8	7 08/37	' 08/27	ﾔ 00/D4	ｬ 00/AC	7E/00	-1	08	88
9	8 09/38	(09/28	ﾕ 00/D5	ｭ 00/AD	7F/00	-1	09	89
10	9 0A/39	) 0A/29	ﾖ 00/D6	ｮ 00/AE	80/00	-1	0A	8A
11	0 0B/30	0 B/00*	ﾜ 00/DC	ｦ 00/A6	81/00	-1	0B	8B
12	- 0C/2D	= 0C/3D	ﾎ 00/CE	-1	82/00	0C/1F	0C	8C
13	^ 0D/5E	‾ 0D/7E	ﾍ 00/CD	-1	83/00	-1	0D	8D
14	¥ 7D/5C	| 7D/7C	ｰ 00/B0	-1	-1	7D/1C	7D	FD
17	q 10/71	Q 10/51	ﾀ 00/C0	-1	10/00	10/11	10	90
18	w 11/77	W 11/57	ﾃ 00/C3	-1	11/00	11/17	11	91
19	e 12/65	E 12/45	ｲ 00/B2	ｨ 00/A8	12/00	12/05	12	92
20	r 13/72	R 13/52	ｽ 00/BD	-1	13/00	13/12	13	93
21	t 14/74	T 14/54	ｶ 00/B6	-1	14/00	14/14	14	94
22	y 15/79	Y 15/59	ﾝ 00/DD	-1	15/00	15/19	15	95
23	u 16/75	U 16/55	ﾅ 00/C5	-1	16/00	16/15	16	96
24	i 17/69	I 17/49	ﾆ 00/C6	-1	17/00	17/09	17	97
25	o 18/6F	O 18/4F	ﾗ 00/D7	-1	18/00	18/0F	18	98
26	p 19/70	P 19/50	ｾ 00/BE	-1	19/00	19/10	19	99
27	@ 1A/40	` 1A/60	ﾞ 00/DE	-1	1A/00*	-1	1A	9A
28	[1B/5B	{ 1B/7B	ﾟ 00/DF	｢ 00/A2	1B/00*	1B/1B	1B	9B
31	a 1E/61	A 1E/41	ﾁ 00/C1	-1	1E/00	1E/01	1E	9E
32	s 1F/73	S 1F/53	ﾄ 00/C4	-1	1F/00	1F/13	1F	9F
33	d 20/64	D 20/44	ｼ 00/BC	-1	20/00	20/04	20	A0
34	f 21/66	F 21/46	ﾊ 00/CA	-1	21/00	21/06	21	A1
35	g 22/67	G 22/47	ｷ 00/B7	-1	22/00	22/07	22	A2
36	h 23/68	H 23/48	ｸ 00/B8	-1	23/00	23/08	23	A3
37	j 24/6A	J 24/4A	ﾏ 00/CF	-1	24/00	24/0A	24	A4
38	k 25/6B	K 25/4B	ﾉ 00/C9	-1	25/00	25/0B	25	A5
39	l 26/6C	L 26/4C	ﾘ 00/D8	-1	26/00	26/0C	26	A6
40	; 27/3B	+ 27/2B	ﾚ 00/DA	-1	27/00*	-1	27	A7

41	: 28/3A	* 28/2A	ケ 00/B9	-1	28/00*	-1	28	A8
42	2B/5D	} 2B/7D	ム 00/D1	」 00/A3	2B/00*	2B/1D	2B	AB
46	z 2C/7A	Z 2C/5A	ツ 00/C2	ッ 00/AF	2C/00	2C/1A	2C	AC
47	x 2D/78	X 2D/58	サ 00/BB	-1	2D/00	2D/18	2D	AD
48	c 2E/63	C 2E/43	ソ 00/BF	-1	2E/00	2E/03	2E	AE
49	v 2F/76	V 2F/56	ヒ 00/CB	-1	2F/00	2F/16	2F	AF
50	b 30/62	B 30/42	コ 00/BA	-1	30/00	30/02	30	B0
51	n 31/6E	N 31/4E	ミ 00/D0	-1	31/00	31/0E	31	B1
52	m 32/6D	M 32/4D	モ 00/D3	-1	32/00	32/0D	32	B2
53	, 33/2C	< 33/3C	ネ 00/C8	、 00/A4	33/00*	-1	33	B3
54	. 34/2E	> 34/3E	ル 00/D9	。 00/A1	34/00*	-1	34	B4
55	/ 35/2F	? 35/3F	メ 00/D2	・ 00/A5	35/00*	-1	35	B5
56	73/5C	_ 73/5F	ロ 00/DB	-1	-1	73/1C	73	F3

番号	下段/上段		Alt	Ctrl	メーク	ブレーク	
15	Back Space	0E/08	B2/00*	B1/00*	0E	8E	
43	Enter	1C/0D	1C/00*	1C/0A*	1C	9C	
44	左Shift	-1	-1	-1	2A	AA	
57	右Shift	-1	-1	-1	36	B6	
58	左Ctrl	-1	-1	-1	1D	9D	
60	左Alt	-1	-1	-1	38	B8	
61	Space	39/20	39/20	39/20	39	B9	
62	右Alt	-1	-1	-1	E0 38	E0 B8	
64	右Ctrl	-1	-1	-1	E0 1D	E0 9D	
75	Insert	52/E0+	A2/00*	92/E0*	E0 52	E0 D2	注1
76	Delete	53/E0+	A3/00*	93/E0*	E0 53	E0 D3	注1
79	←	4B/E0+	9B/00*	73/E0+	E0 4B	E0 CB	注1
80	Home	47/E0+	97/00*	77/E0+	E0 47	E0 C7	注1
81	End	4F/E0+	9F/00*	75/E0+	E0 4F	E0 CF	注1
83	↑	48/E0+	98/00*	8D/E0*	E0 48	E0 C8	注1
84	↓	50/E0+	A0/00*	91/E0*	E0 50	E0 D0	注1
85	Page Up	49/E0+	99/00*	84/E0+	E0 49	E0 C9	注1
86	Page Down	51/E0+	A1/00*	76/E0+	E0 51	E0 D1	注1
89	→	4D/E0+	9D/00*	74/E0+	E0 4D	E0 CD	注1
90	Num Lock	-1	-1	-1	45	C5	
100	*	37/2A	37/00*	96/00*	37	B7	
105	-	4A/2D	4A/00*	8E/00*	4A	CA	
106	+	4E/2B	4E/00*	90/00*	4E	CE	
110	Esc	01/1B	01/00*	01/1B	01	81	
124	Print Screen	-1	-1	72/00	注2参照		
125	Scroll Lock	-1	-1	-1	46	C6	
126	Pause	-1	-1	00/00	注3参照		

注1　キー番号が75,76,79,80,81,83,84,85,86,89に関しては，Shift状態の場合およびNum Lock状態の場合には，メークコードの前と，ブレークコードの後ろに以下のコードが付加されます．

	左Shift状態	右Shift状態	Num Lock状態
メークの前	E0 AA	E0 B6	E0 2A
ブレークの後	E0 2A	E0 36	E0 AA

注2　キー番号が124は，実際には，シフト状態に応じて，以下のようになります．

	通常	Ctrl/Shift状態	Alt状態
メーク	E0 2A E0 37	E0 37	54
ブレーク	E0 B7 E0 AA	E0 B7	D4

注3　キー番号が126は，実際には，シフト状態に応じて，以下のようになります．また，このキーはタイパマティックキーではなく，メークコードしか送信しません．

通常	Ctrl押し下げ状態
E1 1D 45 E1 9D C5	E0 46

・2組の走査コード/文字コードの組み合わせは，前者が標準機能使用時，後者が拡張機能使用時です．

番号	下段/上段		Alt	Ctrl	メーク	ブレーク
95	/	35/2F, E0/2F	A4/00*	95/00*	E0 35	E0 B5 注4
108	Enter	1C/0D, E0/0D	A6/00*	1C/0A, E0/0A	E0 1C	E0 9C

注4　キー番号が95に関しては，Shift状態の場合には，メークコードの前と，ブレークコードの後ろに注1 と同じコードが付加されます．

番号	下段		上段		Alt	Ctrl	メーク	ブレーク
16	Tab	0F/09	バックタブ	0F/00	A5/00*	94/00*	0F	8F
112	F1	3B/00		54/00	68/00	5E/00	3B	BB
113	F2	3C/00		55/00	69/00	5F/00	3C	BC
114	F3	3D/00		56/00	6A/00	60/00	3D	BD
115	F4	3E/00		57/00	6B/00	61/00	3E	BE
116	F5	3F/00		58/00	6C/00	62/00	3F	BF
117	F6	40/00		59/00	6D/00	63/00	40	C0
118	F7	41/00		5A/00	6E/00	64/00	41	C1
119	F8	42/00		5B/00	6F/00	65/00	42	C2
120	F9	43/00		5C/00	70/00	66/00	43	C3
121	F10	44/00		5D/00	71/00	67/00	44	C4
122	F11	85/00*		87/00*	8B/00*	89/00*	57	D7
123	F12	86/00*		88/00*	8C/00*	8A/00*	58	D8

・以下のテーブルは，テンキー部分のキーで，NumLock状態の場合です．NumLockが解除されている場合には，下段と上段が逆になります．

番号	下段		上段		Alt	Ctrl	メーク	ブレーク
91	7	47/37	Home	47/00		77/00	47	C7
92	4	4B/34	←	4B/00		73/00	4B	CB
93	1	4F/31	End	4F/00		75/00	4F	CF
96	8	48/38	↑	48/00		8D/00*	48	C8
97	5	4C/35		4C/00*		8F/00*	4C	CC
98	2	50/32	↓	50/00		91/00*	50	D0
99	0	52/30	Ins	52/00		92/00*	52	D2
101	9	49/39	PgUp	49/00		84/00	49	C9
102	6	4D/36	→	4D/00		74/00	4D	CD
103	3	51/33	PgDn	51/00		76/00	51	D1
104	.	53/2E	Del	53/00	-1	93/00*	53	D3

・以下のキーは，通常かな漢字変換のために予約されていて，プログラムで処理することはできません．

番号	下段		上段		Alt	Ctrl	メーク	ブレーク
1	半角／全角	AF/00*		B0/00*	B2/00*	B1/00*	29	A9
30	英数	B3/00*	Caps Lock	-1	B5/00*	B4/00*	3A	BA
131	無変換	AB/00*		AC/00*	AE/00*	AD/00*	7B	FB
132	変換	A7/00*	前候補	AA/00*	AA/00*	A9/00*	79	F9
133	ひらがな	B6/00*	カタカナ	B9/00*	B9/00*	B8/00*	70	F0

特殊キーの処理

INT 09hルーチン内では，上記のようにキー入力に応じたコードをキー入力バッファにセットしますが，以下の5種類のキー入力に関しては通常のキー入力とは異なり，特殊な処理を実行させます．

[Break]キー

[Ctrl]+[Break]キーを押すと，キー入力バッファをクリアし，BIOSワークエリア（0040h:0071h）に80hを書き込んで，ソフトウェア割り込みINT 1Bhを発生させます．また，キー入力バッファには走査コード・文字コードとも00hが書き込まれます．通常は，INT1Bhのハンドラは，MS-DOSに対して，[Ctrl]+Cキーが押されたように解釈させるDOS内部のBREAKフラグをオンにして，単純にIRETで終了します．したがって，[Ctrl]+Cキーを押したのと同様に，そのBREAKフラグに基づき，ソフトウェア割り込みINT 23hが発生します．これを無効にしたい場合には，リスト3.1のプログラムを使用してください．

リスト3.1　ブレークキーの無効化

```
include         std.inc

                .code

;*****************************************************************************
;*                                                                           *
;*          ブレークキーハンドラの設定                                       *
;*          void        SetBreakHandler(void);                               *
;*          戻り値：    なし                                                 *
;*                                                                           *
;*****************************************************************************
OrgBreakHandler dword   ?

SetBreakHandler proc    uses bx dx ds es
                MSDOS   351Bh                       ;前のハンドラアドレスの保存
                mov     word ptr cs:OrgBreakHandler+2, es
                mov     word ptr cs:OrgBreakHandler, bx
                MOVSEG  ds, cs
                mov     dx,offset cs:BreakHandler
                MSDOS   251Bh                       ;新しいハンドラアドレス
                ret
SetBreakHandler endp

;*****************************************************************************
;*                                                                           *
;*          ブレークキーハンドラの解除                                       *
;*          int         SetBreakHandler(void);                               *
;*          戻り値：    0     ：解除できない                                 *
;*                      0以外：解除できた                                    *
;*                                                                           *
;*****************************************************************************
ResetBreakHandler   proc    uses dx ds
                lds     dx, cs:OrgBreakHandler  ;元のハンドラアドレス
                .if dx != 0                     ;元のアドレスが0でないとき
                    MSDOS   251Bh               ;アドレスを元に戻す
                .else
                    xor     ax, ax              ;元のアドレスが0のとき
                .endif
                ret
ResetBreakHandler   endp

;*****************************************************************************
;*                                                                           *
;*          ブレークキーハンドラ(内部関数)                                   *
;*                                                                           *
;*****************************************************************************
BreakHandler    proc    private
                iret
BreakHandler    endp

                end
```

[Pause]キー

[Pause]キーを押すと，BIOSワークエリア（0040h:0018h）のビット3をオンにして，プログラムの動作を一時中断し，INT 09hルーチン内で他のキー入力を待ちます．この状態ではハードウェア割り込みはすべて通常に処理されますが，割り込み終了後，元のキー入力待ち状態に戻りますので，キー入力があるまでは，実行中のプログラムに制御は戻りません．また，このとき入力したキーコードは捨てられます．

[SysRq]キー

[Alt]+[SysRq]キーを押すと，BIOSワークエリア（0040h:0018h）のビット2をオンにして，EOIを割り込みコントローラに送ります．つぎに，AX=8500hのソフトウェア割り込みINT 15hを発生させます．また，キーを離したときには，AX=8501hとなります．したがって，ユーザープログラムのハンドラルーチンでINT 15hをフックして，AH=85hかどうかを監視すれば，[SysRq]キーに対応する処理ルーチンを作成することができます．

画面印刷

[PrintScreen]（ページ印刷）キーを押すと，PC-9801の[COPY]キーと同様に，キーボード割り込みハンドラはソフトウェア割り込みINT 05hを発生させます．デフォルトのINT 05hハンドラは画面印刷を行います．ただし，グラフィック画面のハードコピーを行うには，DOSに添付されているGRAPHICS.COMを常駐させておく必要があります．このとき，BIOSワークエリアの0050h:0000hに以下の値をセットします．

00h＝画面印刷が行われていない，もしくは前回，正常に終了した．

01h＝画面印刷を実行中．

FFh＝画面印刷を実行中にエラーが発生した．

また，プログラム内で，INT 05hを発行しても，同様に画面印刷が行われます．

リブート

[Ctrl]+[Alt]+[Delete]キーを押すと，コンピュータをリブートさせます．これは，IBM PC系にはリセットボタンがないものがあり，それを代行する意味合いがあります．このとき，BIOSワークエリア（0040h:0072h）に1234hを書き込んで，単純にPOSTルーチンにジャンプします．この1234hはメモリテストをスキップする意味をもっています．

3.2 シフト状態を制御する(AH=02h,12h)

キー入力を行う場合，シフト状況に応じて処理の内容を変えたり，入力する項目にあったシフト状態にしたい場合があります．ここでは，シフト状況の読み取り方法とシフト状態の設定方法に関して解説します．

◎キーボードシフト状況には，シフトキー（[Ctrl],[Shift],[Alt]キー）に関するものと，トグルキー（[CapsLock],[NumLock],[ScrollLock],[Insert]キー）に関するものの2種類があります．

◎IBM PC系のキーボードには大きく分けて，PC 84キーボード系とAT 101キーボード系（これを拡張したOADG 106キーボードなどを含む）があり，シフト状況にも2段階あります．

◎キーボードのシフト状況を読み取るには，2つの方法があります．

(1)　BIOSワークエリアから読み取る方法．

(2)　キーボードBIOSを使用して読み取る方法．

◎キーボードBIOSはBIOSワークのシフト状況を使用して，走査コードから文字コードを生成しているので，このBIOSワークを書き換えることにより，シフト状態を変更することができます．

シフト状況の取得

キーボードシフトの状況は，BIOSワークの0040h:0017h（byte）および0040h:0018h（byte）に入っています（表3.3）．前者がPC 84キーボードがもっていたシフトキーの範囲内を管理し，後者がAT 101キーボードで拡張されたシフトキーに関して管理しています．

また，トグルキー関連に関しては，現在，押し下げ状態かどうかを管理するビットと押し下げごとにトグル状態を変化させるビットの2種類（0040h:0017hのビット4-7と0040h:0018hのビット4-7）があります．このうち，実際にプログラミング上役立つものは，Caps Lock状態とNum Lock状態だけでしょう．

このBIOSワークのシフト状況の値を取り込むためにはリスト3.2の関数を使用します．

キーボードBIOS呼び出しでも，同等の値を読み取ることができます．キーボードBIOSのシフト状況の取得にはシフト状況（AH=02h）とシフト状況（拡張機能）（AH=12h）の2種類があります．後者のほうが上位互換になっており，左右の[Alt]キー，[Ctrl]キーの押し下げ状態が個別に取得できますので，こちらを使用するほうが便利です．

リスト3.3で得られる情報はBIOSワークの値に基づいていますが，一部情報が異なりま

表3.3　BIOSデータエリアのキーボードシフト状況

アドレス	意味			サイズ
0040h:0017h	PC 84キーの範囲のキーボードシフト状況			1バイト
	ビット	意味		
	7	1 = 挿入状態	0 = 挿入状態でない	
	6	1 = Caps Lock状態	0 = Caps Lock状態でない	
	5	1 = Num Lock状態	0 = Num Lock状態でない	
	4	1 = Scroll Lock状態	0 = Scroll Lock状態でない	
	3	1 = Alt(前面)キーが押された		
	2	1 = Ctrl キーが押された		
	1	1 = 左側のShiftキーが押された		
	0	1 = 右側のShiftキーが押された		
0040h:0018h	AT 101拡張キーのキーボードシフト状況			1バイト
	ビット	意味		
	7	1 = Insertキーが押された		
	6	1 = Caps Lockキーが押された		
	5	1 = Num Lockキーが押された		
	4	1 = Scroll Lockキーが押された		
	3	1 = Pause状態		
	2	1 = SysRqキーが押された		
		1 = 左側のAlt(前面)キーが押された		
	0	1 = 左側のCtrlキーが押された		

リスト3.2　BIOSワークエリアのキーボードシフト状況の取得KEYSFT.ASM

```
include         std.inc

bios            segment at 0
                org     0417h                   ; 0040h:0017h
ShiftStatus     word    ?                       ;現在のシフト状況

                .code

;*****************************************************************************
;*                                                                           *
;*          キーボードシフト状況の取得                                       *
;*          unsigned        GetShiftStatusBios(void);                        *
;*          戻り値：        シフト状況                                       *
;*              bit 15   1 = Insert キーが押された                           *
;*              bit 14   1 = Caps Lock キーが押された                        *
;*              bit 13   1 = Num Lock キーが押された                         *
;*              bit 12   1 = Scroll Lock キーが押された                      *
;*              bit 11   1 = Pause状態                                       *
;*              bit 10   1 = SysRq キーが押された                            *
;*              bit  9   1 = 左側のAlt(前面)キーが押された                   *
;*              bit  8   1 = 左側のCtrl キーが押された                       *
;*              bit  7   1 = 挿入状態           0 = 挿入状態でない           *
;*              bit  6   1 = Caps Lock 状態     0 = Caps Lock状態でない      *
;*              bit  5   1 = Num Lock 状態      0 = Num Lock状態でない       *
;*              bit  4   1 = Scroll Lock 状態   0 = Scroll Lock状態でない    *
;*              bit  3   1 = Alt(前面)キーが押された                         *
;*              bit  2   1 = Ctrl キーが押された                             *
;*              bit  1   1 = 左側のShift キーが押された                      *
;*              bit  0   1 = 右側のShift キーが押された                      *
;*                                                                           *
;*****************************************************************************
GetShiftStatusBios  proc    uses ds
                xor     ax, ax
                mov     ds, ax
                assume  ds:bios
                mov     ax, ShiftStatus         ;シフト状況の読み取り
                ret
GetShiftStatusBios  endp

                end
```

リスト3.3　キーボードシフト状況の取得KEYSFT.ASM

```
include         std.inc

                .code

;*****************************************************************************
;*                                                                           *
;*          キーボードシフト状況の取得                                       *
;*          unsigned        GetShiftStatus(void);                            *
;*          戻り値：        シフト状況                                       *
;*              bit 15   1 = SysRq キーが押された                            *
;*              bit 14   1 = Caps Lock キーが押された                        *
;*              bit 13   1 = Num Lock キーが押された                         *
;*              bit 12   1 = Scroll Lock キーが押された                      *
;*              bit 11   1 = 右側のAlt(前面)キーが押された                   *
;*              bit 10   1 = 右側のCtrl キーが押された                       *
;*              bit  9   1 = 左側のAlt(前面)キーが押された                   *
;*              bit  8   1 = 左側のCtrl キーが押された                       *
;*              bit  7   1 = 挿入モード                                      *
;*              bit  6   1 = Caps Lock モード                                *
;*              bit  5   1 = Num Lock モード                                 *
;*              bit  4   1 = Scroll Lock モード                              *
;*              bit  3   1 = Alt(前面)キーが押された                         *
;*              bit  2   1 = Ctrl キーが押された                             *
;*              bit  1   1 = 左Shift キーが押された                          *
;*              bit  0   1 = 右Shift キーが押された                          *
;*                                                                           *
;*****************************************************************************
GetShiftStatus  proc
                KEYBORD 12h                     ;シフト状況の読み取り
                ret
GetShiftStatus  endp

                end
```

す．実際には，[Pause]キーが押されて，プログラム停止状態であることと[Insert]キーが押し下がられているといった2つの情報がBIOSワーク側に多くなっています．ただし，プログラム停止状態では，ハードウェア割り込みしか処理されないため，ハードウェア割り込みを利用した常駐プログラム（TSR）やデバイスドライバでしか使用できません．また，[Insert]キーに関しては，ほとんどの場合，このBIOSワークの情報ではなく，入力キーコードによって，プログラム内で判断するものが多く，使用する意味がないと思われます．

リスト3.3の関数を使用して，シフトキーの押し下げを判定したい場合には，

```
if ( GetShiftStatus() & 0x03 ) {
    シフトキー押し下げ時の処理
}
```
というようにコーディングします．左右別々に判定したい場合には，0x03のかわりに，0x02または0x01を使用します．また，[Ctrl]キーの判定の場合は，
```
if ( GetShiftStatus() & 0x04 ) {
    [Ctrl]キー押し下げ時の処理
}
```
というようになります．

シフト状態の制御

キーボードハードウェア割り込みを処理するプログラムは，BIOSワークエリアにあるシフト状況を見て，走査コードを文字コードに変換しています．また，かな漢字変換プログラム（日本語フロントエンドプロセッサ，FEPと呼ばれる場合もあります）なども同様の処理を行っています．したがって，プログラム内より，このBIOSワークエリアの値を変更することにより，シフト状況を変更することができます．

とくに前述のシフト状況のうち，0040h:0017hのビット4-7は現在のトグルキーによるシフト動作状況を表しています．通常のプログラムの場合，挿入状況はこのBIOSワークエリア内のフラグを参照するものはほとんどなく，[Insert]キーのキー入力コードによりプログラム内で判定しているものが大半です．そのため，このビットを参照する意味はありません．

しかし，ビット6-4のシフト動作状況は，キーボードのLEDインディケータと結びついており，この値を変更することにより，シフト動作状況を変更するとともに，ROM BIOSが自動的にキーボード上のLEDインディケータを更新します．とくに，[NumLock]キーは不用意に押すと，テンキー部分が数値入力キーとカーソル制御キーとで切り替わり，混乱を与える場合がありますので，あらかじめある特定の値にするようにしておくとプログラムの操作性が向上する場合があります．CapsLock状態も，同様に変更することもできます．しかし，特殊な場合を除いては，通常，ユーザーの操作に任せておいて，プログラム内で大文字・小文字変換をさせるほうがわかりやすいと思います．

リスト3.4　キーボードシフト状態の変更KEYSFT.ASM

```
include         std.inc

bios            segment at 0
                org     0417h                       ; 0040h:0017h
ShiftStatus     word    ?                           ;現在のシフト状況

                .code

;*****************************************************************************
;*                                                                           *
;*          キーボードシフト状況の変更                                       *
;*          unsigned     SetShiftStatusBios(unsigned status, int flag);      *
;*          パラメータ：status  シフト状況                                   *
;*                              ONまたはOFFにしたいビットをオンにしておく    *
;*                      flag    0 = OFFにする   1 = ONにする                 *
;*          戻り値：    更新前のシフト状況                                   *
;*                                                                           *
;*****************************************************************************
SetShiftStatusBios  proc    uses ds, status:word, flag:word
                xor     ax, ax
                mov     ds, ax
                assume  ds:bios
                mov     ax, ShiftStatus         ;シフト状況の読み取り
                push    ax
                .if flag == 0                   ;ビットがONのフラグをOFFにする
                    not     status
                    and     ax, status
                .else                           ;ビットがONのフラグをONにする
                    or      ax, status
                .endif
                mov     ShiftStatus, ax
                pop     ax
                ret
SetShiftStatusBios  endp

                end
```

また，あまりお薦めはしませんが，他のシフトキー状況（[Shift],[Alt],[Ctrl]キー）も変更することもできます．たとえば，プログラム内からリブートしたい場合などは，BIOSワークの[Ctrl]キーと[Alt]キーのビットの部分をオンにしておいて，後述のキー入力バッファへの書き込みを利用して，[Delete]キーを書き込み，BIOSワークを元の状態に戻せば実行されます．ただし，プログラムからシフト状態をオンにした場合，プログラム内で明示的にオフにするか，または，実際にシフトキーに対するキーブレーク信号がこないとオフにはなりませんので注意してください．

漢字シフトの制御

アプリケーションプログラムを作成する場合，漢字シフト状況を制御したい場合があります．DOS/Vで標準で添付されるかな漢字変換プログラムには連文節変換プログラム（IBMMKK）があります．このIBMMKKは，日本語入力支援システム（$IAS.SYS）を使用することが必須となっています．この$IAS.SYSの仕様に準拠しているかな漢字変換プログラムには，IBMMKK，ATOK7，WXII+があります．

もうひとつ，標準的なかな漢字変換プログラム制御インターフェースとして，MS-KANJIインターフェースがあります．一応，この2種類を押さえておけば，DOS/Vで使用できるかな漢字変換プログラムの大半が制御できます．

$IAS.SYS

$IAS.SYSは，キーボード入力を拡張し，全角文字の入力をサポートする部分です．この，$IAS.SYSの役割としては，キーボードシフトの管理，ローマ字変換処理およびタスク切り替えの整合性を保っています．

キーボードシフトの管理は，半角の状態も管理できるため，MS-KANJIインターフェースより使いやすいものです．しかし，ローマ字変換を$IAS.SYS中にもち込んでいるため，常駐サイズが大きいことが問題点としてあげられます．最近のかな漢字変換プログラムのなかで，ローマ字変換をもっていないかな漢字変換プログラムはIBMMKKだけですし，こういった無駄を省いて，サイズを小さくしてもらいたいと思っています．

いままでのように，アプリケーション側がいちいち個別にかな漢字変換プログラムに対応していくというのは，もう時代遅れで，今後は，$IAS.SYSのような汎用のかな漢字変換プログラム制御機能が必要でしょう．ただし，$IAS.SYSのもつ機能のうち一般に公開されている機能は，シフト状況の管理機能だけで，MS-KANJIインターフェースのような，アプリケーションプログラムから日本語変換の制御が行えるかな漢字変換モジュールとのインターフェースはありません．

$IAS.SYSを使用している場合は，キーボードBIOS呼び出しを使用することにより，キーボードシフト状態を制御できます．

キーボード状況モードの読み取り

KEYBORD 1301h　　2バイト文字セットのキーボード状況モードの読み取り

[戻り値]

DH =　　予約済み

DL =　　状況モード

ビット7　= 0　非漢字モード
　　　　= 1　漢字モード
ビット6　= 0　ローマ字オフ
　　　　= 1　ローマ字オン
ビット5〜3　　予約済み
ビット2,1 = 00　英数シフト
　　　　= 01　カタカナシフト
　　　　= 10　ひらがなシフト
　　　　= 11　予約済み
ビット0　= 0　半角
　　　　= 1　全角

ただし，$IAS.SYSが組み込まれていない場合には，デフォルトのシフト状況として，DX=0が返ってきます．また，かな漢字変換プログラムが組み込まれていない場合には，DLのビット7はつねに0です．

キーボード状況モードの設定 ――――――――――――――――――――――――

DL =　　状況モード

ビット7　= 0　非漢字モード
　　　　= 1　漢字モード
ビット6　= 0　ローマ字オフ
　　　　= 1　ローマ字オン
ビット5〜3　　予約済み
ビット2,1 = 00　英数シフト
　　　　= 01　カタカナシフト
　　　　= 10　ひらがなシフト
　　　　= 11　予約済み
ビット0　= 0　半角
　　　　= 1　全角

KEYBORD 1300h　2バイト文字セットのキーボード状況モードの設定

また，$IAS.SYSを使用して，漢字モードに入るためには，以下のキーボードBIOSを使用して画面最下行のキーボードシフト状況表示を表示状態にしてください．このシフト状況表示が消去状態の場合には，漢字モードに入ることはできません．この機能と同等の機能がビデオBIOSにもありますが，本書では解説していません．

AL =　00h　シフト状況の表示
　　　01h　シフト状況の消去
　　　02h　シフト状況の表示の有無を読み取る

KEYBORD 14h

[戻り値]

AL =　　　AL=02hの場合に返されます

00h　表示状態

01h　消去状態

この機能も$IAS.SYSが拡張したものであり，$IAS.SYSが組み込まれていない場合には，BIOSはつねにAL=01hを返します．したがって，シフト状況の表示を行った結果が，AL=01hの場合には，$IAS.SYSが組み込まれていないと判断できます．

MS-KANJI --

もうひとつのかな漢字変換インターフェースであるMS-KANJIインターフェースは，かな漢字変換システムとアプリケーションプログラムとの間のソフトウェアインターフェースを規定するものです．このMS-KANJIインターフェースを使用することにより，漢字シフト状況の制御やプログラム内でかな漢字変換を行うことが可能となります．そのため，いろいろなプログラムインターフェースをもっていますが，ここではそのうち，機能コード5（KKMode：かな漢字入力モードのオン／オフの参照と設定）を使用して，漢字シフト状況のオン／オフを制御する方法について解説します．

MS-KANJIインターフェースを使用する場合には，特定のデバイス名MS$KANJIでファイルをオープンし，I/Oコントロールの入力要求（AX=4402h，I/Oリクエストコマンド=3,INT21h）でMS-KANJIインターフェースを呼び出すためのコールアドレスを取得します．MS$KANJIがオープンできない場合には，MS-KANJIインターフェースが組み込まれていないと判断できます．

MS-KANJIインターフェースにパラメータを渡す場合には，以下の構造体のアドレスを渡して，コールします．

Funcparm.wFunc	2バイト	ファンクションの番号
Funcparm.wMode	2バイト	漢字入力モードの参照/設定フラグ
Funcparm.lpKkname	4バイト	かな漢字変換システム名用構造体へのポインタ
Funcparm.lpDataparm	4バイト	データパラメータ用の構造体へのポインタ
Funcparm.Reserved[4]	4バイト	拡張用（つねに0にしておく）

ただし，漢字シフトの制御を行う場合には，後ろの3つのパラメータは使用しませんので，00hをセットしておきます．

現在のかな漢字変換プログラム状況の参照 ------------------

Funcparm.wFunc = 5

Funcparm.wMode = 0 漢字入力モードの参照

　　MSB　ビット15 = 0（参照を意味します）

Funcparmのfarアドレスをスタックにpushする

call　　KKfunc　　MS-KANJIインターフェースの呼び出し

　　（ただし，Pascal形式の呼び出しですので，スタックは

　　kkfunc内で元の状態に戻します）

[戻り値]

AX　　　　　　　=−1 の場合：異常終了

　　　　　　　　=0 の場合：正常終了

Funcparm.wMode　　= 現在の漢字入力モード

　　　LSB　ビット0 = 漢字入力モードがOFFのとき1

　　　　　　ビット1 = 漢字入力モードONのとき1

　　　　　　ビット2 = 変換位置が画面最下行のとき1

　　　　　　ビット3 = 変換位置がカーソル位置のとき1

　　　　　　ビット4 = 0（予約済み）

　　　　　　¦　　　¦

　　　　　　ビット14 = 0（予約済み）

　　　MSB　ビット15 = 0（参照を意味します）

　　　正常終了時，ビット0，1は，いずれか一方が1,他方が0となります．

　　　　　　　　ビット2，3は，いずれか一方が1,他方が0となります．

かな漢字変換プログラム状況の設定 ------------------------

Funcparm.wFunc = 5

Funcparm.wMode = 　漢字入力モードの設定フラグ

　　　LSB　ビット0 = 漢字入力モードをOFFに設定する場合1にします

　　　　　　ビット1 = 漢字入力モードをONに設定する場合1にします

　　　　　　ビット2 = 変換位置を画面最下行に設定する場合1とします

　　　　　　ビット3 = 変換位置をカーソル位置に設定する場合1とします

　　　　　　ビット4 = 0（予約済み）

　　　　　　¦　　　¦

　　　　　　ビット14 = 0（予約済み）

　　　MSB　ビット15 = 1（設定を意味します）

call　　KKfunc　　　　　MS-KANJIインターフェースの呼び出し

[戻り値]

AX　　　　　　　=−1の場合：異常終了

　　　　　　　　=0の場合：正常終了

呼び出し時に，Funcparm.wModeのビット0，1を同時に1にしてはいけません．ビット0，1を同時に0にした場合には，漢字入力モードは変わりません．また，ビット2，3を同時に1にしてはいけません．ビット2，3を同時に0にした場合には，変換位置は変わりません．

この機能を使用した，かな漢字変換プログラムのオン/オフを制御するプログラムをリスト3.5にのせておきます．

FepOn関数を呼び出すことにより漢字モードに，FepOff関数を呼び出すことにより非漢字モードにすることができます．ただし，かな漢字変換プログラムを自動判別していますので，FepOn，FepOff関数を呼び出す前に，必ず，CheckFep関数を呼び出してください．

リスト3.5

```
include          std.inc

Funcparm         struct
    wFunc        word    ?
    wMode        word    ?
    lpKkname     dword   0
    lpDataparm   dword   0
    Reserved     dword   0
Funcparm         ends

                 .code

;***********************************************************************
;*                                                                     *
;*         FEPの種類の調査                                              *
;*         int          CheckFep(void);                                *
;*         戻り値：     FEPの種類   0 = $IAS.SYS                        *
;*                                  1 = MS-KANJI                       *
;*                                 -1 = FEPが組み込まれていない         *
;*                                                                     *
;***********************************************************************
IasDevName       byte    '$IBMAIAS',0            ; $IAS.SYSのデバイス名
MsKanjiDevName   byte    'MS$KANJI',0            ; MS-KANJIのデバイス名
MsKanjiAPIEntry  dword   0                       ; MS-KANJIのエントリアドレス
FepType          word    -1                      ; FEPの種類
Funcbuf          Funcparm                        ; MS-KANJI制御構造体

CheckFep         proc    uses bx cx dx ds
                 MOVSEG  ds, cs
                 assume  ds:@code

                 mov     dx,offset cs:IasDevName
                 MSDOS   3D00h                   ;デバイスのオープン
                 .if !carry?                     ; $IAS.SYSが組み込まれている
                     mov     bx,ax
                     MSDOS   3Eh                 ;デバイスのクローズ
                     xor     ax, ax
                 .else

                     mov     dx,offset cs:MsKanjiDevName
                     MSDOS   3D00h               ;デバイスのオープン
                     .if !carry?                 ; MS-KANJIが組み込まれている
                         mov     bx,ax
                         mov     dx,offset cs:MsKanjiAPIEntry
                         mov     cx,4
                         MSDOS   4402h           ; IOCTL読み込み
                         push    ax
                         MSDOS   3Eh             ;デバイスのクローズ
                         pop     ax
                         .if ax == 4
                             mov     ax, 1
                         .else
                             mov     ax, -1
                         .endif
                     .endif
                 .endif
                 mov     FepType, ax
                 ret
CheckFep         endp

;***********************************************************************
;*                                                                     *
;*         漢字シフトオン                                               *
;*         void         FepOn(void);                                   *
;*         戻り値：     なし                                            *
;*                                                                     *
;***********************************************************************
FepOn                proc    uses dx ds
                     MOVSEG  ds, cs
                     assume  ds:@code

                     .if FepType == 0              ; $IAS.SYS用
                         KEYBORD 1402h             ;シフト状況の表示の有無の取得
                         .if al == 01h             ;消去状態
                             KEYBORD 1400h         ;シフト状況を表示する
                         .endif
                         .if al == 0               ; $IAS.SYSが組み込まれていない
                             KEYBORD 1301h         ;キーボード状況モードの読み取り
                             .if !(dl & 80h)       ;漢字モードでない
                                 or      al, 80h   ;漢字モード
                                 .if !(dl & 40h)   ;ローマ字オフ
                                     or      dl, 05h ;ひらがな・全角
                                 .endif
                                 KEYBORD 1300h     ;キーボード状況モードの設定
                             .endif
                         .endif
                     .elseif FepType == 1          ; MS-KANJIの場合
                         mov     Funcbuf.wFunc, 5  ; FEPのオン/オフ制御
                         mov     Funcbuf.wMode, 0  ;参照
                         call    KKfunc
                         mov     ax, Funcbuf.wMode
                         .if ax & 01h                          ; FEPがオンでないとき
                             mov     Funcbuf.wFunc, 5          ; FEPのオン/オフ制御
                             and     FuncBuf.wMode, 0FFFEh     ; FEPオフビットを消す
                             or      Funcbuf.wMode, 8002h      ; FEPをオンに設定
                             call    KKfunc
                         .endif
                     .endif
                     ret
FepOn                endp

;***********************************************************************
;*                                                                     *
;*         漢字シフトオフ                                               *
;*         void         FepOff(void);                                  *
;*         戻り値：     なし                                            *
;*                                                                     *
;***********************************************************************
FepOff               proc    uses dx ds
                     MOVSEG  ds, cs
                     assume  ds:@code
                     .if FepType == 0              ; $IAS.SYS用
                         KEYBORD 1402h             ;シフト状況の表示の有無の取得
                         .if al == 00h             ;表示状態
                             KEYBORD 1301h         ;キーボード状況ﾓｰﾄﾞの読み取り
                             .if dl & 80h          ;漢字モードでない
                                 and     al, 7Fh   ;漢字モードビットをｵﾌにする
                                 KEYBORD 1300h     ;キーボード状況モードの設定
                             .endif
                         .endif
                     .elseif FepType == 1          ; MS-KANJIの場合
                         mov     Funcbuf.wFunc, 5  ; FEPのオン/オフ制御
                         mov     Funcbuf.wMode, 0  ;参照
                         call    KKfunc
                         mov     ax, Funcbuf.wMode
                         .if !(ax & 01h)                       ; FEPがオンのとき
                             mov     Funcbuf.wFunc, 5          ; FEPのオン/オフ制御
                             and     FuncBuf.wMode, 0FFFDh     ; FEPオンビットを消す
                             or      Funcbuf.wMode, 8001h      ; FEPをオフに設定
                             call    KKfunc
                         .endif
                     .endif
                     ret
FepOff               endp

;***********************************************************************
;*                                                                     *
;*         MS-KANJI制御機能呼び出し関数（内部使用）                     *
;*                                                                     *
;*         MS-KANJIコールアドレス（セグメント値）が0でない場合          *
;*         のみ呼び出します                                             *
;*                                                                     *
;***********************************************************************
KKfunc               proc    near private
                     .if word ptr MsKanjiApiEntry+2 != 0
                         push    cs                    ;パラメータ構造体のｾｸﾞﾒﾝﾄ
                         mov     ax, offset Funcbuf
                         push    ax                    ;パラメータ構造体のｵﾌｾｯﾄ
                         call    MsKanjiAPIEntry       ; MS-KANJIの呼び出し
                     .else
                         mov     FuncBuf.wMode, 0      ;取り敢えず0にしておく
                     .endif
                     ret
KKfunc               endp

                     end
```

3-3 キーボードからの入力 (AH=00h,01h,10h,11h)

キーボードからの入力は，DOSレベルで行う場合とキーボードBIOSを使用して行う場合の2通りがあります．ここでは，キーボードBIOSを使用した文字の読み取り，文字の入力状況の確認方法，キーリピート速度の変更方法に関して解説します．

◎DOSレベルの入力の場合，[F1]キーなどの機能キーは，一文字目に00hが読み込まれ，続けて2文字目を読み込むことにより，どの機能キーかが判断できます．

◎キーボードBIOSを使用すれば走査コードも取得できるため，フルキー部分の文字か，テンキー部分の文字かが判断できます．

◎キーボードBIOSでも漢字などのDBCS文字を読み込めます．

◎キーボードBIOSでの次文字の読み取り，文字の入力状況には，PC 86キー用の標準機能とAT 101キーボード用の拡張機能の2種類があります．

◎キーリピート速度は標準では遅めに設定されています．また，キーボードBIOSや外部コマンドにより変更することができます．

◎BIOSワークエリア上のキー入力バッファは，SBCS文字しか書き込まれず，日本語環境で読み取られる文字は，DBCSコードジェネレータ内のキー入力バッファが使用されますので，基本的には操作してはいけません．

次文字の読み取りと文字の入力状況

キー入力に関しては，DOSレベルで行うよりも，キーボードBIOSを使用するのが便利です．また，PC-9801と違い，IBM PC系のFEPの場合には必ずINT 16hをサポートしていますので，漢字なども問題なく処理できます．したがって，普通IBM PC系専用のプログラムはほとんどキーボードBIOSを使用しています．だたし，DOSのキー入力ではキーボードのかわりにファイルからデータを読み込めるようにリダイレクトできるのに対して，キーボードBIOSではそのようなことはできません．

キーボードBIOSを使用したキー入力に関しては，ソフトウェア割り込みINT 16h を使用します．このうち，キー入力に関しては，以下の4種類があります．

AH =	00h	次文字の読み取り
	01h	文字の入力状況
	10h	次文字の読み取り（拡張機能）
	11h	文字の入力状況（拡張機能）

AH=00h,10hのほうは，キー入力があるまで待つのに対して，AH=01h,11hは，データが

表3.4 **走査コード・文字コードの組み合せ**

種　類	上位バイト	下位バイト
SBCS文字	走査コード	文字コード
SBCS文字(カタカナ)	00h	文字コード
拡張コード	走査コード	00hまたはE0h
ALT+JIS8ビット入力	00h	文字コード
DBCS文字の第1バイト	00h	第1バイト
DBCS文字の第2バイト	00h	第2バイト

キー入力バッファ内にあるかどうかを調べるだけで，キー入力バッファからは取り出しません．また，拡張機能のほうは，AT　101キーボードで拡張されたキーに対応していますので，そちらのほうを使用したほうが，[F11]キーや[F12]キーなどまで問題なく使用できます．

4種類とも，読み取る値は走査コード（AH）と文字コード（AL）の対になった値が取り込まれます（表3.4）．また，[F1]～[F12]や[Home]キーなどの特殊キーは，文字コードに00hまたはE0hがセットされます．漢字などの2バイトコードの場合には，走査コードに00hがセットされます．

文字コードを処理する場合には，まず，拡張コードかどうかを判断するために，下位バイトが00hかE0hかをチェックします．ただし，通常文字のなかで，DBCS文字の第2バイトにもE0hは存在するため，E0hの場合は，上位バイトが00hかどうかをチェックして，00hならDBCS文字の第2バイト目だと判断してください（DOS/Vプログラミングには直接関係はありませんが，J3100 ATOKでは，漢字の場合上位バイトとしてFEh，FFhが得られますので，走査コードが00hかF0h以上なら，DBCS文字の第2バイト目と判断すれば安全です）．

これ以外の文字は，通常文字ですので，とくにフルキーとテンキーを区別しない場合には，走査コードは無視してもかまいません．また，漢字コードはすべてシフトJISコードに対応しています．

また，このようなDBCS文字の第2バイト目と拡張コードの判断を避けるため，また拡張コードやDBCS文字を扱いやすくするためには，リスト3.6の関数を使用すると便利です．

リスト3.6の関数では，次文字の読み取り（拡張機能）を行い，まず拡張キーコードかどうかを判断して（ALが00hかE0hの場合），拡張コードの場合には共通的に操作コード部分にDBCS文字に存在しない80hを設定しています．つぎに，文字コードがDBCS文字の第1バイト目かを判定して，DBCS文字の第1バイト目の場合には，続けて，次文字の読み取り（拡張機能）を実行して，第2バイト目を読み込み，高位バイトに第1文字目を，下位バイトに第2文字目をセットします．これ以外はSBCS文字ですので，高位バイトに00hをセットしています．

この関数を使用することにより，SBCS文字，DBCS文字とも一度の呼び出しで取得することができます．また，[F1]などの特殊キーも0x80??というコードがセットされますので，C言語側でも，高位バイトが00h（SBCS文字），80h（拡張コード），それ以外（DBCS文字）というように，簡単に判断が行えるはずです．

同じキーを押し続ける操作をするプログラムのキー入力ルーチンの場合，キー入力を行ってしまうと，もし別のキーが押された場合に，そのキーコードを保管しておく必要があります．そういった場合，読みたいキーコードを指定して，そのキーであれば入力を行う

リスト3.6　　**キーボード入力処理KEYIN.ASM**

```
include         std.inc

                .code

;*****************************************************************
;*                                                               *
;*        キーボード入力処理                                      *
;*        uint       GetKb(void);                                *
;*        パラメータ:なし                                         *
;*        戻り値:      キー入力値                                 *
;*                                                               *
;*****************************************************************
GetKb           proc    uses bx
                KEYBORD 10h                     ;次文字の読み取り(拡張機能)
                .if ah != 0 && ( al == 0 || al == 0E0h )
                    mov     al, ah              ;拡張キーコード
                    mov     ah, 80h
                .else
                    .if (al >=  81h && al <=  9Fh) || ¥
                        (al >= 0E0h && al <= 0FCh)
                        mov     bl, al              ;漢字の場合
                        KEYBORD 10h
                        mov     ah, bl
                    .else
                        xor     ah, ah              ;通常の半角
                    .endif
                .endif
                ret
GetKb           endp

                end
```

リスト3.7　　**キーコード指定によるキーボード入力処理KEYIN.ASM**

```
include         std.inc

                .code

;*****************************************************************
;*                                                               *
;*        キーコード指定によるキーボード入力処理                  *
;*        uint       GetKey(uint keycode);                       *
;*        パラメータ:  keycode  キー入力対象コード               *
;*        戻り値:      キー入力値                                 *
;*                                                               *
;*****************************************************************
ShiftKeyStatus  word    ?

GetKey          proc    uses bx, keycode:word
                mov     bx, keycode             ;入力対象コード
                .if bx != 0                     ;コード指定あり
                    KEYBORD 11h                 ;文字の入力状況
                    .if zero?
                        xor     ax, ax          ;バッファーにない
                    .else
                        .if ah != 0 && ( al == 0 || al == 0E0h )
                            mov     al, ah      ;拡張キーコード
                            mov     ah, 80h
                        .endif
                        .if ax != bx            ;入力対象コード以外
                            xor     ax,ax
                        .endif
                    .endif
                .else
                    mov     ax, -1              ;常時キー入力する
                .endif
                .if ax != 0
                    KEYBORD 12h                 ;シフト状況の読み取り
                    mov     cs:ShiftKeyStat, ax

                    KEYBORD 10h                 ;次文字の読み取り
                    .if ah != 0 && ( al == 0 || al == 0E0h )
                        mov     al, ah          ;拡張キーコード
                        mov     ah, 80h
                    .else
                        .if (al >=  81h && al <=  9Fh) || ¥
                            (al >= 0E0h && al <= 0FCh)
                            mov     bl, al      ;漢字の場合
                            KEYBORD 10h
                            mov     ah, bl
                        .else
                            xor     ah, ah      ;通常の半角
                        .endif
                    .endif
                .endif
                ret
GetKey          endp

;*****************************************************************
;*                                                               *
;*        キーボードシフト状況の取得                              *
;*        uint       GetShiftStatus2(void);                      *
;*        戻り値:      シフト状況                                 *
;*                                                               *
;*****************************************************************
GetShiftStatus2 proc
                mov     ax, cs:ShiftKeyStat     ;シフト状況
                ret
GetShiftStatus2 endp

                end
```

ようなキー入力ルーチンがあれば便利です.

リスト3.7の関数では，キーコードが指定されていない場合には，前記キーボード入力処理と同じ処理をしています．また，キーコードが指定されている場合には，まずキー入力状況を確認し，指定されたキーコードが入力されている場合にのみ，次文字の読み取り処理を行っています．たまっていない場合には，0でリターンするようにしています.

また，シフトキーの読み取りも，別途関数を用意するのではなく，文字読み取り時点でシフト状況を取得して，ワークエリアに書き込み，シフト取得関数では，そのワークエリアからシフト状況を読み取るようにしています．厳密にいうと異なるケースもあるのですが，シフト状況をキー入力を行うのと同時に保存することにより，あとでシフト状況を読み取った場合に，キー入力を行った際のシフト状況と異なる可能性を避けるためです.

キーリピート速度の変更（AX=0305h）

PC86キーボードは，タイパマティックと呼ばれる，キーリピート機能をもっています．このキーリピート速度は，通常，遅めに設定されています（デフォルトの値は，AT互換機系の場合1秒間に10.0文字，PS/2系の場合は1秒間に10.9文字に設定されています．また，

リスト3.8 **キーボードタイプ速度の設定KEYIN.ASM**

```
include         std.inc

                .code

;****************************************************************************
;*                                                                          *
;*      キーボードタイプ速度の設定                                          *
;*      void        SetTypematicRate(int rate, int delay);                  *
;*      パラメータ：rate    タイプ間隔時間                                  *
;*                  delay   遅延時間（0-3）                                 *
;*                          = 0    250ms                                    *
;*                          = 1    500ms                                    *
;*                          = 2    750ms                                    *
;*                          = 3   1000ms                                    *
;*      戻り値：    なし                                                    *
;*                                                                          *
;*   タイプ間隔時間(1秒当たりにタイプできる文字数)                          *
;*                                                                          *
;*    | 設定値 | 速度 | 設定値 | 速度 | 設定値 | 速度   |                    *
;*                                                                          *
;*    | 0      | 30.0 | 11     | 10.9 | 22     | 4.3    |                    *
;*    | 1      | 26.7 | 12     | 10.0 | 23     | 4.0    |                    *
;*    | 2      | 24.0 | 13     | 9.2  | 24     | 3.7    |                    *
;*    | 3      | 21.8 | 14     | 8.6  | 25     | 3.3    |                    *
;*    | 4      | 20.0 | 15     | 8.0  | 26     | 3.0    |                    *
;*    | 5      | 18.5 | 16     | 7.5  | 27     | 2.7    |                    *
;*    | 6      | 17.1 | 17     | 6.7  | 28     | 2.5    |                    *
;*    | 7      | 16.0 | 18     | 6.0  | 29     | 2.3    |                    *
;*    | 8      | 15.0 | 19     | 5.5  | 30     | 2.1    |                    *
;*    | 9      | 13.3 | 20     | 5.0  | 31     | 2.0    |                    *
;*    | 10     | 12.0 | 21     | 4.6  | 32-255 | 予約済 |                    *
;*                                                                          *
;*                                                                          *
;****************************************************************************
SetTypematicRate    proc    uses bx, rate:byte, delay:byte
                mov     bl, rate                ; タイプ間隔時間
                mov     bh, delay               ; 遅延時間
                KEYBORD 0305h                   ; キーボードタイプ速度の設定
                ret
SetTypematicRate    endp

                end
```

遅延時間は両者とも500ミリ秒になっています）．このキーリピート速度は，リスト3.8の関数を使用して，プログラム的に速くすることができます．Delayで指定された時間以上同じキーが押し続けられると，Rateで指定された速度でキーリピートします．すなわち，Rate=0，Delay=0を指定すると，0.25秒待ったあとで，1秒間に30文字の反復速度でキーリピートするわけです．

しかし，キーボード操作に関しては，好みや慣れなどもあり，キーリピート速度を速くすることを望まない人もいます．したがって，以下のコマンドを使用して，自分の望むキーリピート速度に設定することをお薦めします．Rateは32からSetTypematicRate関数（リスト3.8）の値を引いた値を，Delayの値は同関数の値＋1を指定します．

```
MODE CON: RATE=32 DELAY=1
```

この設定で，最高速度のキーリピートとなります．ただし，問題点としては，キーリピートが速くなりますので，そのキーリピート間隔内にスクロールを実行させないと，キー入力バッファがあふれてしまい，ビープ音が鳴りだしてしまいます．このため，スクロールが遅くなってしまいます．話題の比較広告のスクロールでDOS/Vマシンが遅かったのも，このビープ音を鳴らせないために，一番遅い機種（386SX 12MHz）に合わせて，逆にキーリピート間隔をソフト的に長めにセットしているといった点を考慮しなければなりません．

このビープ音に関しては，PC-9801のようにBIOSワークエリアに特定の値をセットすることにより鳴らさなくするようなことはできません．したがって，適切なタイミングで，キー入力バッファを消去してしまうしかありません．たとえば，継続してスクロール要求がある場合などは，スクロールごとにキー入力バッファを消去するようにすれば，[↑]，[↓]キーを離した瞬間にスクロールが止まりますし，逆に操作性も十分あがります．キー

リスト3.9 **キーボードバッファの消去処理KEYIN.ASM**

```
include         std.inc

                .code

;****************************************************************************
;*                                                                          *
;*      キーボードバッファの消去処理                                        *
;*      void        KeyBufClear(void);                                      *
;*      戻り値：    なし                                                    *
;*                                                                          *
;****************************************************************************
KeyBufClear     proc
                .while 1
                    KEYBORD 11h                 ;キー入力状況(拡張機能)
                    .break .if zero?            ;キー入力が行われていない
                    KEYBORD 10h                 ;次文字の読み取り(拡張機能)
                .endw
                ret
KeyBufClear     endp

                end
```

入力バッファを消去するためには，リスト3.9の関数を使用します．

リスト3.9の関数は，キー入力状況を見て，キー入力があった場合には，次文字の読み取りでキー入力バッファから読み取り，そのデータを捨てる作業を繰り返しているだけです．

3-4 キー入力バッファへの書き込み（AH=05h）

PC-9801などでは，よくBIOSワークエリア上のキー入力バッファを直接更新して，あたかもキー入力があったように見せかけることがあります．DOS/Vでは，プログラミングで，キーボードから入力されたものと同様に，キー入力バッファに走査コードと文字コードを書き込むことができます．

◎キーボードBIOSに，キー入力バッファへのデータの書き込み機能があります．

◎この機能を使用すれば，BIOSワークエリア上のキー入力バッファなのか，DBCSコードジェネレータ上のキー入力バッファなのかを気にする必要はありません．

キーボードBIOSの機能コード（AH=05h）は，あたかもキーボードから入力されたのと同様に，プログラムからキー入力バッファに書き込むことができます．ただし，キー入力バッファ自身は15文字分しか使用できませんので，大量の文字を書き込むことはできません．また，キーコードにない文字も書き込むことができますので，常駐プログラムと親プログラム間の連絡や特殊なキー入力にも使用できます．

この機能を使用すれば，親プログラムから子プログラムを呼び出す際に，DOSのプログラム起動呼び出しを使用するのではなく，キー入力として書き込み子プログラム名+Enterを書き込めば，COMMAND.COMがキー入力と判断して，子プログラムを起動してくれます．これにより，余計なバッチ領域が不要になったり，親プログラム自身のコード部分が開放されるために，より大きなメモリ空間を使用することができます．ただし，親プログラムの呼び出し元がCOMMAND.COMまたは同等機能をもつシェルでないと使いません．

これを応用すれば，カット&ペースト用の常駐プログラムやメニューなどいろいろ使いみちはあると思います．

リスト3.10　キー入力バッファへのデータの書き込みPUTKEY.ASM

```
include         std.inc

                .code

;**************************************************************************
;*                                                                        *
;*      キーボードバッファへのデータの書き込み                            *
;*      int         PutKeyBuffer(uint keycord);                           *
;*      パラメータ：keycord 走査コード+文字コード                         *
;*      戻り値：    0 = 機能は正常に終了                                  *
;*                  1 = キーボードバッファに空きがない                    *
;*                                                                        *
;**************************************************************************
PutKeyBuffer    proc    uses cx, keycode:word
                mov     cx, keycode             ; ch = 走査コード, cl = 文字コード
                KEYBORD 05h                     ; キーボードバッファへの書き込み
                xor     ah, ah                  ; 上位バイトをクリア
                ret
PutKeyBuffer    endp
                end
```

3-5 走査コードの変換 (AH=4Fh,INT 15h)

常駐プログラムのホットキーやFEPよりも先に処理したいキーがある場合などに便利な，ROM BIOSサービスルーチンが用意されています．通常なら，INT 09hのキーボードハードウェア割り込みをフックして，自分で処理することも可能ですが，キーボードのハードウェアに関しての知識が必要で，なかなか難しいものがあります．

◎キーボードハードウェア割り込みハンドラ（ハードウェア割り込みINT 09h）は，キー入力があると，ALに走査コードをセットして，キーボードインターセプト（AH=4Fh INT 15h）を呼び出します．

◎キーボードインターセプトルーチン内で走査コードを変更したり，処理を行ったりすることができます．

◎この割り込みはハードウェア割り込み中から呼び出されるため，通常BIOS割り込みやDOSのシステムコールを使用することはできません．

キーボードインターセプトに渡される走査コードは表3.2（走査コードセット1）に記載したメークブレークコードです．複数バイトの走査コードの場合には1バイトずつ渡されますので，処理ルーチン内部でフラグ管理などによって，連続した走査コードであることを管理しなければいけません．このキーボードインターセプトルーチン内では，押されたキーコードを変更したり，自分自身で処理することができます．また，この割り込みをフックした場合，自分で走査コードを処理しない場合には，フックする前のアドレスに制御を渡す必要があります．

キーボードインターセプトルーチンが呼び出された場合，ALに走査コードがセットされています．キーボードインターセプトルーチンでは以下の3種類の処理のうち，どちらかを実行します．

(1) 走査コードを処理しない場合には，そのまま後続の処理ルーチンに制御を渡します（すなわち，フックする前のINT 15hの差すアドレスへfar jmpします）

(2) 走査コードを変更する場合には，ALを新しい走査コードに置き換え，スタック上のキャリーフラグをオンにセットします．ROM BIOSにそのまま処理させたい場合には，IRETで終了します．後続のルーチンにさらに処理させたい場合には，後続の処理ルーチンに制御を渡します．

(3) 走査コードを内部的に処理した場合には，スタック上のキャリーフラグをオフにして，IRETで終了します．INT 09hルーチンは，ALの走査コードを処理しません．

表3.2の走査コードセット1は，IBM 5576-A01型キーボードでの走査コードです．他のキーボードの場合の走査コードを確認できるように，リスト3.11のサンプルプログラムを添

リスト3.11　キーボードインターセプトルーチンに渡される走査コード

```
;*******************************************************************************
;***                                                                         ***
;***        File Name         :  SCANKEY.ASM                                ***
;***        Description       :  キーボードの走査コードの確認                 ***
;***                                                                         ***
;*******************************************************************************
                .model  tiny                        ; .COMファイルの生成

                include     std.inc

                .code

                org 100h

start:          MSDOS   3515h                       ;元のINT 15hのアドレス取得
                mov     word ptr OrgSystemIntr+2, es ;割り込みアドレスの保存
                mov     word ptr OrgSystemIntr, bx
                mov     dx, offset cs:SystemInter   ;新しいINT 15h割り込み
                                                    ;  ハンドラーのアドレス
                MSDOS   2515h                       ;割り込みアドレスの書き換え
                .repeat
                    KEYBORD 0                       ;キー入力待ち
                .until al == 01Bh                   ; ESCキーが押されたら終了
                lds     dx, cs:OrgSystemIntr        ;元のINT 15hの割り込み
                MSDOS   2515h                       ;  ハンドラに戻す
                MSDOS   TERMINATE,00h               ;プログラム終了

OrgSystemIntr   dword   ?                           ;割り込みアドレス保存用

SystemInter     proc                                ;新しい割り込みハンドラ
                pushf                               ;フラグの保存
                .if ah == 4Fh                       ;キーインターセプト時
                    PUSHM   <ax,cx>
                    push    ax
                    mov     cl, 4                   ;上位4ビットの処理
                    shr     al, cl
                    call    showHex                 ; 16進数文字の出力
                    pop     ax
                    and     al, 0Fh                 ;下位4ビットの処理
                    call    showHex                 ; 16進数文字の出力
                    mov     al, ' '
                    VIDEO   0Eh                     ;ブランクの出力
                    VIDEO   0Eh                     ;ブランクの出力
                    POPM    <cx,ax>
                .endif
                popf                                ;フラグの復元
                jmp     cs:OrgSystemIntr            ;元の割り込みルーチンに制御を渡す
SystemInter     endp

showHex         proc
                .if al >= 10                        ; '0'から'F'に変換
                    add     al, 'A'-10
                .else
                    add     al, '0'
                .endif
                VIDEO   0Eh                         ;上位4ビット文字の出力
                ret
showHex         endp

                end start
```

リスト3.12　キー走査コードを処理する

```
SystemInter     proc                                ;新しい割り込みハンドラ
                pushf                               ;フラグの保存
                .if ah == 4Fh                       ;キーインターセプト時
                    .if al== 02h                    ; 1が押された
                        PUSHM   <ax,bx>             ;レジスタの保存
                        push    es
                        xor     bx,bx
                        mov     es,bx
                        mov     bl,es:[417h         ; BIOSワークのシフト状況
                        pop     es
                        .if bl & 08h                ; Altキーが押し下げ状態

                            ;ここで本体のフラグなどをオンにしたりする
                            ;ような，走査コードに関連する処理を行います．

                            POPM    <bx,ax>         ;レジスターの復元
                            popf                    ;フラグの復元
                            push    bp
                            mov     bp, sp
                            and     word ptr [bp+6], not 0001h
                                                    ;スタック上のキャリーフラグを
                                                    ;  オフにする
                            pop     bp
                            iret                    ;走査コードを処理したため
                                                    ;後続のハンドラに制御を渡さない
                        .endif
                        POPM    <bx,ax>
                    .endif
                .endif
                popf                                ;フラグの復元
                jmp     cs:OrgSystemIntr            ;元の割り込みルーチンに制御を渡す
SystemInter     endp
```

付します．このプログラムは，走査コードを表示するだけで，処理・変更は一切行いません．また，[Esc]キーを押すことにより終了します．

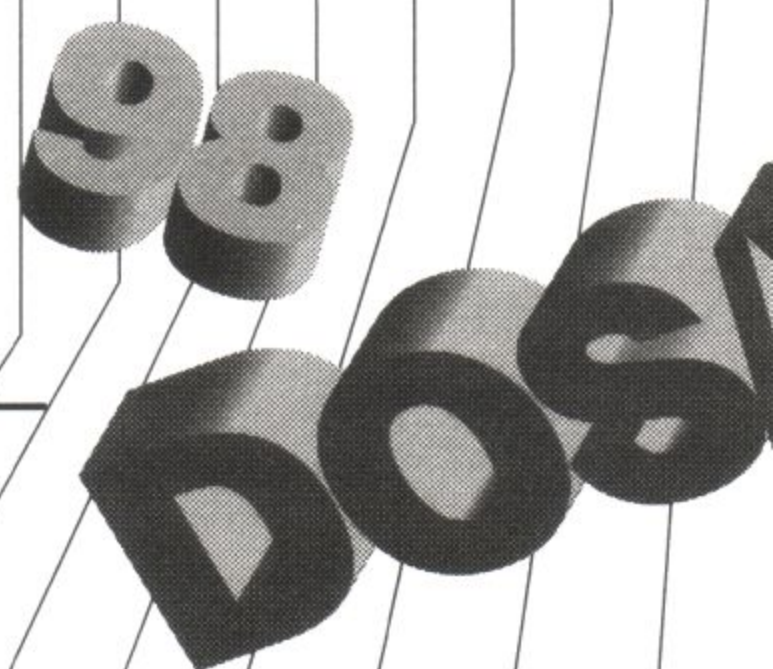
つぎに，走査コードを処理する例を見てみます．キーインターセプト処理のキーとなる部分は，スタック上のキャリーフラグをオフにして，後続のハンドラに制御を渡さないように，IRETで終了している点です．リスト3.12のサンプルプログラムでは，コメントになっていますが，中央部分に実際に走査コードを処理する部分を付け加えます．

リスト3.12の例では，[Alt]+1が押されたときの処理例ですが，FEPとして，WXIIなどを使用すると，このキーコードはプログラムに渡される前にFEPによって処理されてしまいます．しかし，中央処理部分でプログラム本体のキー入力フラグなどをオンにしてやることにより，後続のFEPには処理されずに，プログラム本体ではそのキー入力フラグを見て処理を行えばよくなります．

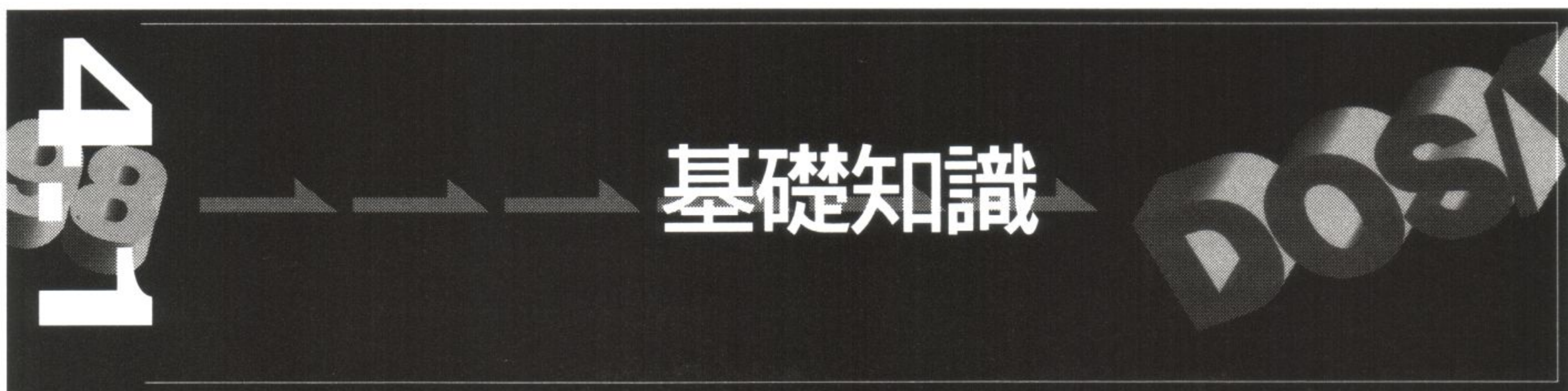
第4章

98からDOS/Vの世界へ

DOS/V

98 DOS/V

この章では，IBM PC/AT系および PS/55系で使用できるRS-232C割り込みプログラミングに関して解説します．しかし，IBM PC/AT系のRS-232C関連BIOSにはほとんど機能がなく制約も多いため，サンプルソースとともに，直接，非同期通信用のLSIを制御した通信環境について説明を行います．

RS-232C編

4-1 基礎知識

IBMのマニュアルを見ると，ASYNC入出力とか非同期通信と書かれている場合がよくあります．このASYNC入出力／非同期通信というのは，多分IBM用語だと思われるのですが，通常RS-232Cによるシリアル通信と呼ばれるもので，本来シリアル通信は1文字ごとに同期をとっていますので，非同期通信という呼び方はおかしいと思っています．また，別名として調歩同期式通信と呼ばれる場合もあります．したがって，これ以降はシリアル通信と呼ぶことにします．

PC-9801では，通信制御用のLSIとしては，非同期通信と同期通信の両者をサポートしているインテル社の8251A（および同等品）が使用されていますが，IBM PC系では，非同期通信専用のナショナルセミコンダクタ社のINS8250B（または，同等品のNS16450および上位互換のNS16550）が使用されています．このLSIは同期通信はできませんが，そのぶん非同期通信に関しては，8251Aと比較すると機能が豊富です．そのため，同期通信を行いたい場合には，別途同期通信専用のボードが必要となります．

一般に，AT互換機の場合は2個のシリアルポートを，IBM純正機種では1個のシリアルポートを搭載しています．これは，AT互換機の場合には，マウスとしてバスマウスではなく，シリアルマウスを使用している場合が多く，通常COM1をマウスに，COM2をシリアル通信に使用しているからです．このため，DOS/Vで通信関連プログラムを作成する場合には，通信ポートを指定できるようにしておかなければなりません．

また，シリアルポートはBIOS設計上はシリアルポートを最大4つまで搭載できるように設計されています．ただし，一般的に使用されるのは，COM1，COM2であり，COM3，COM4に関してはシリアルボードのディップスイッチやジャンパスイッチで設定するようになっている場合が多く，ボードによって異なっています．

ハードウェア割り込み

DOS/Vでのシリアル通信BIOS（INT 14h）は，受信割り込みを行っておらず，機能が少ないため，最近の高速通信には使用できません．また，実際にあまり使用されていません．そこで，シリアルポートを直接制御したプログラムを作成するわけですが，このためにはハードウェア割り込み機能を使用しなければなりません．このハードウェア割り込みを制御しているのが，8259Aと呼ばれる割り込みコントローラで，Programable Interrupt Controlerを略してPICと呼ばれることもあります．ここでは，8259Aを使用した割り込み処理に関して解説を行います．

表4.1				8259AのI/Oポートアドレス
	R/W	マスタ	スレーブ	
ICW1	W	0020h	00A0h	初期設定コマンドワード1
OCW2	W			動作制御用コマンドワード2
OCW3	W			動作制御用コマンドワード3
IRR	R			割り込み要求レジスタ
ISR	R			インサービスレジスタ
ICW2	W	0021h	00A1h	初期設定コマンドワード2
ICW3	W			初期設定コマンドワード3
ICW4	W			初期設定コマンドワード4
OCW1	W			動作制御用コマンドワード1
IMR	R			割り込みマスクレジスタ

また，AT互換機での割り込み一覧は，資料編4にのせていますので，参考にしてください．

ハードウェア割り込みは，実際にはシリアル通信以外にタイマやキーボードなどの多くの周辺機器で使用されていますが，通常は各BIOSがそれらを覆い隠しており，一般的にシリアル通信以外では使用することはまれです．また，ハードウェア割り込みにはNMI（Non Maskable Interrupt）と呼ばれる割り込みもありますが，この割り込みに関しても，通常プログラムで使用することはありませんので，ここではシリアル通信に限って解説を行います．

AT互換機やPS/2，PS/55では，マスターとスレーブの2つの8259Aを搭載しています．それぞれ，表4.1に示すI/Oポートに割り当てられています．

ハードウェア割り込みの手順

ハードウェア割り込みが発生すると，現在実行中のプログラムは中断され，以下のような手順で割り込みハンドラに制御が移ります．

(1) 割り込みコントローラから割り込みベクトルを受け取る．

(2) スタックにフラグを待避する．

(3) 割り込み禁止状態にする．

(4) スタックにCS,IPレジスタの値を待避する

(5) 割り込みベクタの指すアドレスに制御を渡す．

したがって，割り込みハンドラは通常のソフトウェア割り込みと同様に，IRETで終了すれば問題はありません．ただし，ハードウェア割り込みの優先順位や再割り込みを避けるために，割り込みコントローラを使用して制御しなければなりません．したがって，通常の割り込みハンドラは以下のようなコーディングになるはずです．

(1) 全レジスタをスタックに待避する．

(2) 上位の割り込みを許可する．

(3) 割り込み処理を行う．

(4) 全割り込みを禁止する．

(5) EOIコマンドを実行する．

(6) スタックから全レジスタを復帰させる．

(7) IRETで終了する．

IRETが実行されると，さきほど中断したプログラムに制御が戻されます．

ハードウェア割り込みはいつ動作するかわからないため，基本的に，処理内部でDOSのシステムコールやBIOS割り込みを使用することはできません．また，CLI命令を使用すると，他のハードウェア割り込みが停止してしまい，他の周辺機器での割り込みが処理できなくなり，さまざまな問題が生じますので，割り込みを禁止期間の処理もなるべく短時間で終わらせて，STI命令を実行するか，または割り込みハンドラ自身の処理を終らせてください．したがって，ハードウェア割り込みでは，簡単な内部処理だけを行って，プログラム内にフラグを立て，実際の処理はINT 1Ch（タイマ割り込み），INT 28h（DOSキー入力アイドル待ち）などの割り込み中に処理を行うようにしてください．といっても，上記割り込み内でも，きちんとした条件を守らないと，DOSのシステムコールやBIOS機能の使用はできませんので，注意してください．

また，どのような状態で割り込みが発生するかが不明なため，プログラムを作成する場合には「……だろう」という気持ちではなく，きちんとすべての環境を整える必要があります．

よく間違えるものとして，スタックのオーバーフローとCPUのフラグレジスタの方向フラグがあります．すなわち，ハードウェア割り込みが発生した場合に，スタックエリアはどこのものを使用しているか不明ですし，その容量もわかりません．したがって，ハードウェア割り込みでスタックを比較的大量に消費する場合は，割り込み処理の先頭で，自分のプログラム内部のスタックに切り替えておくべきでしょう．また当然のことながら，終了時には元のスタックに戻しておく必要があります．この場合，スタックを固定にしていると割り込み処理中に再度ハードウェア割り込みが発生した場合に，さきほどの内部スタックを壊してしまうことがありますので，再入する可能性がある場合には，必ず再入対策を行ってください．

もうひとつの問題点の方向フラグに関しては，割り込み発生時点で方向フラグが昇順・降順のどちらになっているかわからないため，バイト命令（MOVSB,STOSBなど）やワード命令（MOVSW,LODSWなど）などの方向フラグによって動作が変化する命令を使用する前には，CLD，STD命令を使用して，必ず方向フラグを初期化してください．

また，通常はないと思いますが，ハードウェア割り込みベクタをソフト的に呼び出した場合には，そのままではハードウェア割り込みと区別がつかないため，割り込み処理の先頭で，ISRの該当ビットがセットされているかをチェックすれば，ハードウェア割り込みと内部割り込みを区別することができます．

関連ハードウェア

DOS/V系のシリアル通信制御用のLSIとしては，前述したように3種類のチップが使用されています，これらのシリアル通信コントローラは以下の機能を提供しています，

◎スタートビット，ストップビット，パリティビットの付加および除去

◎文字モードにおける完全二重バッファリング

◎プログラム可能なボーレイトジェネレータ（PBRG）による50～38400bps程度の

動作

◎データ長，ストップビット長，パリティビットの形状の設定

◎送信・受信・エラー・回線状況による割り込みの制御

◎回線ブレークの生成と検出

◎モデム制御機能

また，プログラミング上は基本的には大きく2種類と考えて問題ありません．3種類のチップはすべてINS8250Bとして動作が可能ですので，INS8250Bとしてコーディングを行えばとくに問題はありません．また，NS16550には高速転送時に有利なFIFOモードをもっています．したがって，コーディング方法としては単純にINS8250Bとしてコーディングを行うか，またはNS16550かどうかをチェックし，NS16550であればFIFOモードで，それ以外のチップであればINS8250Bとしてコーディングする方法です．判定方法に関しては，初期化のところで解説します．

また，PC-9801の場合はシリアル通信を行う場合に，8251だけでなく8253，8255，8259の4つの周辺LSIを制御することが必要となりますが，INS8250B系はシリアル通信に関して機能が豊富ですので，このチップだけで通信速度，データビット数，RS-232C信号線の制御，割り込み条件の設定などの制御が可能です．したがって，シリアル通信に関しては，INS8250B系のLSIの操作だけで大半の処理が済んでしまいます．あとは通常のハードウェア割り込み処理のために，8259割り込みコントローラに対する割り込みマスクの設定やEOIの送出だけです．

また，これらのシリアル通信コントローラ内のレジスタへのアクセスは8バイトの連続したI/Oポートを使用して行われます．また，前述したとおり，最大4つまでのシリアルポートを実装できますので，IBM PC系ではPOST処理中にシリアルポートの実装をチェックし，BIOSワークエリアにそれぞれのI/Oポートの先頭アドレスを書き込みます．シリアルポートが実装されていない場合は，BIOSワークエリアには0000hが書き込まれます．

図4.1 シリアルポートコントローラブロック図

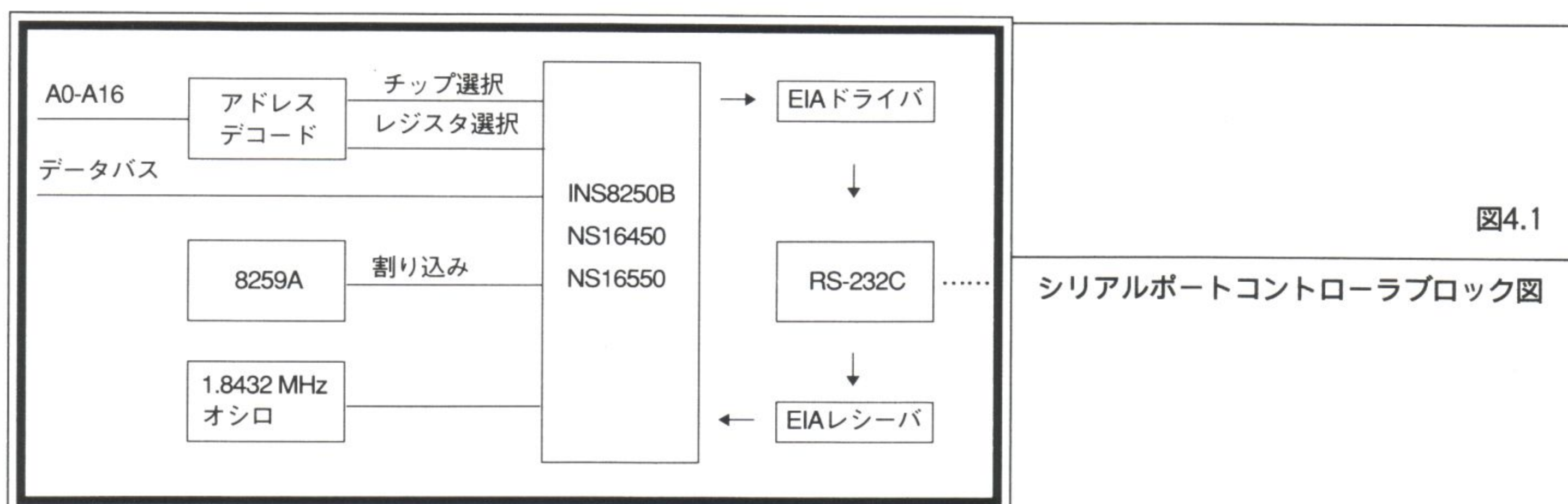



表4.2 シリアル通信I/Oポートベースアドレス

アドレス	意味	サイズ
0040h:0000h	COM1のI/Oポートのベースアドレス	1ワード
0040h:0002h	COM2のI/Oポートのベースアドレス	1ワード
0040h:0004h	COM3のI/Oポートのベースアドレス	1ワード
0040h:0006h	COM4のI/Oポートのベースアドレス	1ワード

表4.3				シリアル通信I/Oポートアドレス
ポートアドレス	DLAB	入出力	機能(8250B系)	機能(NS16550)
0nF8h(*1)	0	IN	受信データレジスタ	受信FIFOバッファ
		OUT	送信保持レジスタ	送信FIFOバッファ
	1	I/O	ボーレート分周レジスタLSB	ボーレート分周レジスタLSB
0nF9h	0	I/O	割り込み許可レジスタ	割り込み許可レジスタ
	1	I/O	ボーレート分周レジスタMSB	ボーレート分周レジスタMSB
0nFAh		IN	割り込み識別レジスタ	割り込み識別レジスタ
		OUT	------------------------	FIFO制御レジスタ
0nFBh		I/O	ライン制御レジスタ	ライン制御レジスタ
0nFCh		I/O	モデム制御レジスタ	モデム制御レジスタ
0nFDh		IN	ラインステータスレジスタ	ラインステータスレジスタ
0nFEh		IN	モデムステータスレジスタ	モデムステータスレジスタ
0nFFh		I/O	スクラッチパッドレジスタ	スクラッチパッドレジスタ

*1：一般に，COM1は03F8h，COM2は02F8h，COM3は03E8h，COM4は02E8hに割り当てられています．ただし，BIOSワークエリアにあるベースアドレスを参照すること．

したがって，通信プログラムを書く際には，このBIOSワークアリアを参照して，シリアル通信コントローラのI/Oポートのベースアドレスを取得して，そこにアクセスするようにしてください．すなわち，一般的にはCOM1が03F8h，COM2が02F8hの場合が多いのですが，ボードの設定などにより変更することが可能ですので，ポートアドレスを固定してしまってコーディングを行うことは避けてください．シリアル通信コントローラのレジスタが割り当てられているアドレスは，表4.3を参照してください．また一部の特殊なシリアルボードに関しては，ここで記載した内容と異なる場合がありますが，その場合にはボードに添付のマニュアルを参照してください．

また，これらのレジスタに連続してIN命令，OUT命令でアクセスする場合には，シリアル通信コントローラが追いつかないことがありますので，通常は時間稼ぎのためにダミーの命令（JMP $+2など）を入れます．しかし，一般的には途中にいろいろな命令が入り，連続してアクセスすることも少ないので，それほど気にすることはありませんが，安心のために適度なウェイトを入れておくとよいでしょう．

割り込みベクタ

IBM PC系では，通常は割り込みベクタは以下の2つしか用意されていません．

◎ 0Bh COM2/COM4割り込み（IRQ3）

◎ 0Ch COM1/COM3割り込み（IRQ4）

しかし，ATバス系の割り込みはエッジトリガになっているため，1つの割り込みベクタで同時に2つの通信ポートをサポートすることはできません．そこで，通常はCOM1，COM2をそれぞれIRQ3，IRQ4に割り当てて，COM3，COM4を使用していない別のIRQに割り当てられるようなジャンパスイッチをもっているシリアル通信ボードが多く存在しています．

そこで，割り込みベクタに関してはデフォルトの値をもっておいて，指定があった場合にはその割り込みベクタを使用するようにすればよいでしょう．

4.2 初期化と終了処理

シリアル通信を行う場合には，FIFOバッファモード／文字モードの選択および通信速度，データビット長，ストップビット長，パリティの種類を設定しなければなりません．また，COM1からCOM4に対応して，制御用のI/Oポートのアドレスの取得，割り込みベクタの設定も個別に行う必要性があります．

ここでは，シリアルポートに応じた割り込み処理の開始，終了処理およびシリアル通信コントローラの初期化と各種通信パラメータの設定に関して解説を行います．

◎NS16550の場合には，FIFOモードで，それ以外の場合には文字モードで動作するようにします．

◎シリアル通信コントローラのI/Oポートのベースアドレスは BIOSワークエリアより取得します．

◎割り込みベクタに関しては，デフォルトの値をもっておき，指定があった場合にはその割り込みベクタを指定するようにします．

初期化手順としては，以下の手順になります．

割り込みベクタおよびI/Oポートのベースアドレスの決定

まず，シリアル通信で使用するシリアルポート番号から，使用する割り込みベクタおよびI/Oポートアドレスを決めます．ただし基礎知識で説明したとおり，I/Oポートのベースアドレスは BIOSワークエリアから取得できますが，割り込みベクタは標準のCOM1，COM2に関しては決まっていますが（変更することもできるシリアル通信ボードもあります），COM3，COM4に関しては決まっていませんので，外部から指定できるようにして

リスト4.1　I/Oポートのベースアドレスと割り込み番号の決定

```
/*****************************************************************************/
/*                                                                           */
/*                        これは，関数の一部分です                              */
/*                                                                           */
/*        PortBase は，I/Oポートのベースアドレス                                */
/*        channel  は，シリアルポート   1=COM1, 2=COM2, 3=COM3, 4=COM4          */
/*        irqNo    は，IRQ番号          -1=ﾃﾞﾌｫﾙﾄ, 0-15=明示的指定             */
/*                                                                           */
/*****************************************************************************/


// シリアルポートのI/OベースアドレスをBIOSワークエリアから取得する．

    int far *pbase = (int far *)0x00000400L;

    PortBase = *(pbase + channel - 1);       // I/Oポートのベースアドレスの取得


// 割り込み番号が指定されていない場合には，ポート番号により，IRQ番号を決定する．

    if (irqNo == -1) {                      //割り込み番号が指定されていないとき
        if ( channel == 1 || channel == 3 )      // COM1 と COM3 は IRQ3
            irqNo = 3 ;
        else                                     // COM2と COM4 は IRQ2
            irqNo = 2 ;
    }


// IRQ番号により，実際の割り込み(INT)番号と8259の割り込みマスクを求める．

    if (irqNo <= 7) {                            //マスタ8259の場合
        RsInt = irqNo + 0x08;                    //実際の割り込み(INT)番号
        MaskRsInt = 1 << irqNo;                  // 8259 割り込みマスク
        ImrPort = 0x21 ;                         //マスタIMRのアドレス
    }
    else {                                       //スレーブ8259の場合
        RsInt = irqNo + 0x70;                    //実際の割り込み(INT)番号
        MaskRsInt = 1 << (irqNo - 8);            // 8259 割り込みマスク
        ImrPort = 0xA1 ;                         //スレーブIMRのアドレス
    }
```

表4.4 **FIFO制御レジスタ**

ビット	内　容
7,6	受信FIFOレジスタ割り込み用トリガレベル
	00 =　1バイト
	01 =　4バイト
	10 =　8バイト
	11 = 14 バイト
5-3	つねに0
2	1 = 送信FIFOバッファのリセット
1	1 = 受信FIFOバッファのリセット
0	0 = 文字モード
	1 = FIFOモード

リスト4.2 **NS16550の判定IS16550.ASM**

```
include         std.inc

                .code

;*******************************************************************************
;*                                                                             *
;*          NS16550の判定                                                       *
;*          int         isNS16550(int port)                                    *
;*          パラメータ：port     ポート番号(1-4)                                 *
;*          戻り値：      0 = NS16550でない                                      *
;*                       16 = NS16550AN                                        *
;*                       -1 = シリアルポートが実装されていない                    *
;*                                                                             *
;*******************************************************************************
isNS16550       proc    uses bx dx, port:word

; FIFO制御レジスターのアドレスを求める

                mov     ax, port                ;ポート番号
                dec     ax                      ; 0-3に変換
                shl     ax, 1                   ;ワードポインタ
                mov     bx, ax
                push    es
                mov     ax, 40h
                mov     es, ax                  ; BIOSワークのセグメント
                mov     dx, word ptr es:[bx]    ;ポートレジスタアドレス
                add     dx, 02h                 ;一般的にDXは以下の値になる
                                                ; COM1   COM2  COM3  COM4
                                                ; 03FAh, 2FAh, 3EAh, 2EAh
                                                ; DX=FIFO制御ﾚｼﾞｽﾀｱﾄﾞﾚｽ（出力）
                                                ; DX割り込み識別ﾚｼﾞｽﾀｱﾄﾞﾚｽ（入力）
                pop     es

                .if dx != 02h                   ;シリアルポートが実装されているとき
                    in      al, dx              ;現在の値
                    mov     bh, al              ; UARTレジスタの元の値の保管

; FIFOバッファリングモードにしてみる

                    mov     al, 11000001b       ; 14ﾊﾞｲﾄﾊﾞｯﾌｧの開始
                    out     dx, al
                    in      al, dx              ; FIFOバッファがオンになったかを
                                                ;   チェックする
                    and     al, 11000000b       ; FIFOバッファ長だけ取りだす
                    push    ax                  ; FIFOバッファ状態の保管


                    mov     al, bh              ;元のレジスタの値
                    and     al, 11000000b       ; FIFOバッファ長だけ元に戻す
                    .if al == 0C0h              ; 16550ANなら
                        or      al, 00000001b   ; FIFO Enable
                    .endif
                    out     dx, al              ;元のレジスタ状態に戻す

                    pop     ax                  ;割り込み識別レジスタの上位2ビット
                                                ;  は以下の意味を持つ
                                                ;   00 = 16550でない
                                                ;   10 = 16550
                                                ;   11 = 16550AN = 16ﾊﾞｲﾄFIFOﾊﾞｯﾌｧ
                    .if al == 0C0h              ; 16550ANのとき
                        mov     ax,16           ; 16バイトFIFOバッファ
                    .else                       ;それ以外は
                        xor     ax, ax          ;   FIFOバッファなしとして扱う
                    .endif
                .else
                    mov     ax, -1              ;シリアルポートが実装されていない
                .endif
                ret
isNS16550       endp

                end
```

おきます．I/Oポートのベースアドレスと割り込み番号の決定をリスト4.1にのせておきます．

また，シリアル通信コントローラにも，2種類のもの（INS8250B系とNS16550系）があります．NS16550系の場合はFIFOバッファをもっており，文字落ちもほとんどなくなりますので，NS16550系の場合にはFIFOモードで，それ以外の場合はINS8250B系として文字モードでシリアル通信を行うようにプログラミングすべきです．このFIFOモードを制御するのがFIFO制御レジスタで割り込み識別レジスタ（読み出し専用）と同じアドレスにあり，書き込み専用レジスタです（表4.4）．

16550系を判定するためには，リスト4.2の関数を使用します．

割り込みの禁止

シリアル通信コントローラを初期化中に誤動作することを防ぐために，初期化中は割り込みを禁止状態にしておきます．シリアル通信の割り込みを制御するのが割り込み許可レジスタで，4種類の割り込みを個別に制御できます（表4.5）．割り込みを禁止するのですから，シリアルI/Oポートのベースアドレス＋1の位置にある割り込み許可レジスタに0を出力して，すべてのビットをオフにします．

表4.5　割り込み許可レジスタ

ビット	内　容
7–4	つねに0
3	1 = モデムステータス割り込みを許可
2	1 = ラインステータス割り込みを許可
1	1 = 送信レジスタ空き割り込みを許可
0	1 = 受信データレディ割り込みを許可 FIFOモードではタイムアウト割り込みも許可

◎モデムステータス割り込み

CD(DCD), CI(RI), DR(DSR), CS(CTS)信号が最後の読み出し以降に変化したときに，割り込みが発生します．

◎ラインステータス割り込み

フレーミングエラー，パリティエラー，オーバーランエラー，ブレーク信号検出時に割り込みが発生します．

◎送信レジスタ空き割り込み

送信保持レジスタ（文字モード）または送信FIFOバッファ（FIFOモード）が空になったときに割り込みが発生します．

◎受信データレディ割り込み

受信データレジスタにデータがあるか（文字モード），受信FIFOバッファに指定したトリガレベル以上のデータを受信している場合に，割り込みが発生します．また，FIFOモードで，受信FIFOバッファに1文字以上のデータがある場合，4文字分以上の文字間隔の間，次のデータを受信しない場合に，タイムアウト割り込みが発生します．

ライン制御レジスタの設定

シリアル通信を行う場合には，通信速度・キャラクタ長・パリティ・ストップビット長を決定しなければなりません．これらのパラメータを設定するのがライン制御レジスタです（表4.6）．このライン制御レジスタは入出力可能で，シリアルI/Oポートのベースアドレス＋3の位置にあります．

8250B系は自分の内部にボーレートジェネレータをもっており，PC-9801などのように8253タイマICを操作することは不要で，外部クロック信号を内部のボーレート分周レジスタの値で分周して，通信速度を決定します．外部クロック信号としては，IBM PC系ではつねに1.8432MHzが使用されているため，分周値は表4.7のようになります．

したがって，通信速度・キャラクタ長・パリティ・ストップビット長を設定する手順は以下のようになります．

(1)　ライン制御レジスタのビット7のDLABを1にする．

(2)　ボーレート分周レジスタに8ビットずつアクセスする．

(3)　ライン制御レジスタのビット7のDLABを0に戻す．

(4)　(3)と同時に，キャラクタ長・パリティ・ストップビット長を設定する．

コーディングとしては，リスト4.3のようになります．

表4.6 ライン制御レジスタ

ビット	内　容
7	ベースアドレスおよびその次のレジスタの意味を決定 表4.3でのDLAB 0 = 受信データ・送信保持，割り込み許可レジスタ 1 = ボーレート分周レジスタ（LSB，MSBの順）
6	ブレーク制御 0 = 通常動作 1 = ブレーク送出状態
5-3	パリティチェック xx0 = パリティなし 001 = 奇数パリティ 011 = 偶数パリティ 101 = パリティはつねに1 111 = パリティはつねに0
2	ストップビット 0 = 1ストップビット 1 = 2ストップビット ただし，キャラクタ長が5のときには1.5ストップビット
1,0	キャラクタ長 00 = 5ビット 01 = 6ビット 10 = 7ビット 11 = 8ビット

表4.7 通信速度と分周値の関連

通信速度	分周値（10進数）	分周値（16進数）
50	2304	0900h
75	1536	0600h
150	768	0300h
300	384	0180h
600	192	00C0h
1200	96	0060h
2400	48	0030h
4800	24	0018h
9600	12	000Ch
19200	6	0006h
38400	3	0003h
57600	2	0002h
115200	1	0001H

通信速度×分周値×16=1843200が成り立つ

リスト4.3　　通信速度・キャラクタ長などの設定

```
#define LCR     0x03

unsigned int speed_8250[12] =
        {1536, 768, 384, 192,  96,  48,  24,  12,   6,   3,   2,   1};
/*  bps     75  150  300  600 1200 2400 4800 9600 192K 384K 576K 1152 */
/*  speed    0    1    2    3    4    5    6    7    8    9   10   11 */

/***************************************************************************/
/*                                                                         */
/*                      これは，関数の一部分です                           */
/*                                                                         */
/*        PortBase は,I/Oポートのベースアドレス                            */
/*        speed    は,上記 speed_8250へのインデックス                      */
/*        length   は,キャラクタ長   0=5ﾋﾞｯﾄ, 1=6ﾋﾞｯﾄ, 2=7ﾋﾞｯﾄ, 3=8ﾋﾞｯﾄ        */
/*        stops    は,ストップビット  1=1ｽﾄｯﾌﾟﾋﾞｯﾄ, 2=2ｽﾄｯﾌﾟﾋﾞｯﾄ              */
/*        parities は,パリティの種類  0=ﾊﾟﾘﾃｨなし, 1=奇数ﾊﾟﾘﾃｨ, 3=偶数ﾊﾟﾘﾃｨ   */
/*                                                                         */
/***************************************************************************/

    outp(PortBase+LCR, 0x80 );                  // ライン制御レジスタのDLABを1にする
    outp(PortBase, speed_8250[speed]);          // 通信速度にあった分周値の設定(LSB)
    outp(PortBase+1, speed_8250[speed] >> 8);   // 通信速度にあった分周値の設定(MSB)

    rsStatus = (unsigned char)length;           // キャラクタ長
    rsStatus |= ((stops > 1) << 2);             // ストップビット長
    rsStatus |= ((parities & 3) << 3);          // パリティの種類
    outp(PortBase+LCR, rsStatus);               // ライン制御レジスタの設定

                                                                     //
bit76543210

                                                                     //
bit 000PPSLL
```


割り込みベクタの保存と書き換え

シリアル通信のハードウェア割り込みハンドラを登録します．プログラム終了時のために現在の割り込みベクタの値を保存し，新しい割り込みハンドラの値を登録します．登録する割り込み番号はリスト4.1で求めたRsIntです．

FIFOモードの設定と割り込みの許可

NS16550ANの場合には，16バイトのFIFOバッファをもっていますので，FIFOモード

に切り替えます．そうでない場合には，文字モードに切り替えます．FIFOモードを制御するには，表4.4のFIFO制御レジスタを使用します．問題は，受信FIFOバッファの割り込みトリガレベルをいくつにするかだと思われます．FIFOモードで，受信データレディ割り込みが許可されている場合，受信FIFOバッファに受信した文字数が，トリガレベルに達すると，受信データレディ割り込みが発生し，文字数がトリガレベルより下がるとクリアされます．また，1文字以上が受信FIFOバッファにあって，最後の文字を受信するか，受信FIFOバッファから読み出してから，4文字分の時間が経過している場合にタイムアウト割り込みが発生します．

したがって，14バイトですとオーバーランが心配ですし，4バイトではバッファに空きが多すぎると思われますので，8バイトにトリガレベルを設定することにします．

割り込みハンドラ内での詳細な処理は「4.4　送受信処理」で解説しますが，基本的な考え方だけをここでは解説します．

割り込みはプログラムのオーバーヘッドを減らすように，5種類の割り込みを4種類の優先順位に振り分けています．

◎優先順位1　ラインステータス割り込み（ラインエラーとBreak受信）
◎優先順位2　受信データレディ割り込み
◎優先順位2　タイムアウト割り込み（FIFOモードのみ）
◎優先順位3　送信レジスタ空き割り込み
◎優先順位4　モデムステータス割り込み

このうちタイムアウト割り込みは受信データレディ割り込みと同等にみなせるため，4種類の割り込みに関して検討してみます．

ラインステータスに関しては，Break受信に関しては端末側なのでチェックは不要なこと，また，その他のラインエラーに関しては，受信データ時に判定するか，パソコン通信などではMNP方式によって文字化けなどに対応しているため，通常は起きることも非常にまれなので無視することにします．また，モデムステータム割り込みに関しても，送受信時にそれぞれの信号を確認・設定したほうが安全ですので，割り込みで動作させないことにします．

したがって，送信空き割り込みと受信データレディ割り込みだけを割り込み駆動で動作させることにします．しかし，無駄な割り込みを避けるため，送信空き割り込みは，送信データがある場合にだけ発生させるようにこまめに管理していくことにします．

そこで，ここでは，受信データレディ割り込みだけを許可しますが，このとき，モデム制御レジスタのビット3をオンにすることを忘れないでください．これを忘れると割り込みが発生しません．また同時に初期設定が完了しましたので，RS(RTS)信号とER(DTR)信号をオンにします．

割り込みコントローラの割り込みマスクの指定

シリアルポートをハードウェア割り込みで運用するには，割り込みコントローラ8259の

もつ割り込みマスクレジスタ（IMR）の該当部分を0にしなければなりません．通常は割り込みマスクレジスタの操作だけで，8259自体の初期化は不要です．実際には割り込みコントローラにはマスタとスレーブがあり，IRQ0-7とIRQ8-15がそれぞれ対応します．また，実際にはリスト4.1で，ImrPort（8259のIMR用ポートアドレス）と，MaskRsInt（IMR中の該当ビット）を求めていますので，その値を使用して該当ビットをオフにしてください．

このとき，IRQ0，IRQ8がビット7側になりますので，注意してください．すなわち，IRQの順番とIMRのビット順が逆になっており混乱しがちですので注意してください．

修了処理

終了時手順としては，以下の手順になります．

◎割り込みコントローラの割り込みマスクの指定

さきほどと逆の手順で，割り込みコントローラの割り込みマスク情報を変更します．

◎割り込み許可レジスタの値を元に戻す

割り込み許可レジスタの値を，割り込みなし状態，またはプログラムが起動する前の値に戻します．

◎割り込みベクタを元に戻す

最後に割り込みハンドラのアドレスを起動前の値に戻します．

以上の手順をまとめたのが，リスト4.4のrsOpen関数とrsClose関数です．

リスト4.4　RS-232Cオープン処理RSMAIN.C

```
#include    <dos.h>
#include    "common.h"              /* 共通インクルードファイル         */
#include    "rs232c.h"              /* RS-232C用ヘッダファイル          */

int     PortBase ;                  /* シリアルポートI/Oベースアドレス  */
uchar   RsInt ;                     /* 割り込み(INT)ベクタ番号          */
uchar   MaskRsInt ;                 /* 8259 IMR のマスク情報            */
uchar   ImrPort ;                   /* 8259 IMR の I/Oポートアドレス    */

static int far *pbase = (int far *)0x00000400L;
                                    /* I/Oベースアドレステーブルの先頭  */

static uint speed_8250[12]  =        /* 回線速度インデクッスによる分周値 */
        {1536, 768, 384, 192,  96,  48,  24,  12,   6,   3,   2,   1};
/*  bps     75  150  300  600 1200 2400 4800 9600 192K 384K 576K 1152 */
/*  speed    0    1    2    3    4    5    6    7    8    9   10   11 */

static char ChannelSave = 0;        /* オープン済みシリアルポート記憶用 */

/*************************************************************************/
/*                                                                       */
/*          RS-232Cオープン処理                                          */
/*          void        rsOpen(int channel, int speed, int length,       */
/*                              int parities, int stops, int irqNo)      */
/*          パラメータ：channel      シリアルポート   1-4 = COM1-COM4     */
/*                      speed        speed_8250[]へのインデックス        */
/*                                     0=75bps ~ 8=19200bps ~ 11=1152Kbps */
/*                      length       キャラクタ長                        */
/*                                     0=5ビット,1=6ビット,2=7ビット, 3=8ビット */
/*                      parities     パリティの種類                      */
/*                                     0=パリティなし,1=奇数パリティ,3=偶数パリティ */
/*                      stops        ストップビット数  (1,2)             */
/*                      irqNo        IRQ番号   -1=デフォルト, 0-15=明示的指定 */
/*          戻 り 値：なし                                               */
/*                                                                       */
/*************************************************************************/
void rsOpen(int channel, int speed, int length, int parities, int stops, int irqNo)
{
        uchar rsStatus ,
              FifoBufSize ;


    if (ChannelSave != 0)                           //すでにオープンされている場合は
        rsClose();                                  //  一度クローズする
    ChannelSave = channel;                          //オープンしたポート番号を保存する

    PortBase = *(pbase + channel - 1);              // I/O ポートのベースアドレスの取得


// 割り込み番号が指定されていない場合には，ポート番号により，IRQ番号を決定する．

    if (irqNo == -1) {                              //割り込み番号が指定されていないとき
        if ( channel == 1 || channel == 3 )         // COM1 と COM3 は IRQ3
            irqNo = 3 ;
        else                                        // COM2と COM4 は IRQ2
            irqNo = 2 ;
    }


// IRQ番号により，実際の割り込み(INT)番号と8259の割り込みマスクを求める．

    if (irqNo <= 7) {                               //マスタ8259の場合
        RsInt = irqNo + 0x08;                       //実際の割り込み(INT)番号
        MaskRsInt = 1 << irqNo;                     // 8259 割り込みマスク
        ImrPort = 0x21 ;                            //マスタIMRのアドレス
    }
    else {                                          //スレーブ8259の場合
        RsInt = irqNo + 0x70;                       //実際の割り込み(INT)番号
        MaskRsInt = 1 << (irqNo - 8);               // 8259 割り込みマスク
        ImrPort = 0xA1 ;                            //スレーブIMRのアドレス
    }

    FifoBufSize = isNS16550(channel) ;              // FIFO バッファサイズを求める

    outp(PortBase+IER, 0);                          //割り込み禁止

    outp(PortBase+LCR, 0x80 );                      //ライン制御レジスタのDLABを1にする
    outp(PortBase, speed_8250[speed]);              // 回線速度にあった分周値の設置(LSB)
    outp(PortBase+1, speed_8250[speed  >> 8);       // 回線速度にあった分周値の設置(MSB)

    rsStatus = (unsigned char)length;               // キャラクタ長
    rsStatus |= ((stops > 1) << 2);                 // ストップビット長
    rsStatus |= ((parities & 3) << 3);              // パリティの種類
    outp(PortBase+LCR, rsStatus);                   // ライン制御レジスタの設定

    OrgVect = _dos_getvect( RsInt );                // 割り込みベクタの保存
    _dos_setvect( RsInt, rsHandler );               // 新割り込みハンドラの登録

    if ( FifoBufSize != 0 ) {                       // NS16550の場合
```

```
    outp(PortBase+FCR,0x81h)                // FIFOモードの設定
  }

  outp(PortBase+MCR, inp(PortBase+MCR) | 0x0b);
                                          // 8250の割り込み許可
                                          //   RTS,DTRをオンにする
  outp(PortBase+IER, RXE);                //受信割り込み許可

  inp(PortBase+LSR);                      //ライン状況のリセット

  outp(ImrPort, (OrgImr = inp(ImrPort)) & ~MaskRsInt);
                                          //割り込みマスクの指定

}

/****************************************************************************/
/*                                                                          */
/*          RS-232Cクローズ処理                                              */
/*          void        rsClose(void) ;                                     */
/*          パラメータ：なし                                                 */
/*          戻 り 値：なし                                                   */
/*                                                                          */
/****************************************************************************/
void rsClose( void )
{
    if ( ChannelSave != 0 ) {

outp(ImrPort, OrgImr);

outp(PortBase+IER, OrgIer);
        outp(PortBase+MCR, 0);                  // 8250の割り込み禁止
                                                //   RTS,DTRをオフにする

_dos_setvect( RsInt, OrgVect );

ChannelSave = 0 ;
                                                                        }
}
```

4.3 信号線の制御

RS-232C（V.24と呼ばれる場合もあります）とは，本来はモデムとパソコンなどのシステム間を接続するインターフェース（25ピンの信号線）の規約のことです．この25ピンの信号線を使用して通信を行うため，日本では一般的にRS-232Cと呼ぶとシリアル通信のことを指す場合が多いです．このシリアル通信を行う場合，単純にデータを転送するだけではなく，相手システム，モデム，回線を制御するために信号線を制御しなければなりません．ここでは，各信号線とその処理に関して解説します．

◎RS-232Cの信号線は本来25ピンありますが，パソコンでのシリアル通信の場合には，大半が使用されていません．

◎データの送受信はそれぞれSD，RDを使用するため，全二重通信（同時に送受信が可能）ができます．

◎信号線として制御できるのは，出力用ではRS,ER，入力用ではCD,DR,CS,CIの6信号線だけです．

◎信号線の出力はモデム制御レジスタで，信号線の入力はモデムステータスレジスタを使用します．

◎通信をはじめるときにERをオンにします．回線が接続されるとDRがオンになりますので通信が可能となります．また，モデムが相手システムのキャリアを検出するとCDがオンになります．

◎自システムのRSは相手システムのCSに接続されていて，自システムが受信可能のときにRSをオンにします．相手システムはCSがオンであれば送信可能と判断できます．これによりハードウェアフロー制御（RS/CS制御）が可能となります．

◎公衆回線で，相手システムから電話があるとCIがオンになります．ホストシステムなどを作成するときに必要になる場合があります．

RS-232Cには規約で25ピンの信号線がありますが，実際にシリアル通信で使用されるの

表4.8 RS-232Cの制御信号

ピン番号		入出力	略号		名称	機能
8	1	IN	CD	DCD	受信キャリア検出	キャリア検出状態でオン
3	2	IN	RD	RXD	受信データ	受信データ
2	3	OUT	SD	TXD	送信データ	送信データ
20	4	OUT	ER	DTR	データ端末レディ	オフにすると回線回線切断
7	5	---	SG	GND	信号グラウンド	
6	6	IN	DR	DSR	データセットレディ	回線接続状態でオン
4	7	OUT	RS	RTS	送信要求	ハードウェアフロー制御(RS/CS制御) 受信可能時にオンにする
5	8	IN	CS	CTS	送信可	ハードウェアフロー制御(RS/CS制御) オンのときに送信できる
22	9	IN	CI	RI	被呼表示	公衆回線で着信時にオン

ピン番号は前者がD-SUB25ピンの場合，後者が9ピンDシェルコネクタの場合です．

は表4.8に記載したピン番号の信号線だけです．したがってノートパソコンなどでは，パソコン側に9ピンしかないコネクタを用意している場合が多く見受けられます．

モデム制御レジスタ

モデム制御レジスタ（MCR）のビット0と1は，それぞれ出力信号であるER（DTR）とRS(RTS)の制御に使用します．ビット2と3は，それぞれOUT1，OUT2としてユーザーに開放されています．OUT1はHayesコマンド内蔵モデムの電源オンリセット用で，OUT2は8250B/16550シリアルコントローラ割り込み要求のマスクに使用されています．したがって，8250B/16550を割り込みモードで使用する場合には，必ずビット3を1にしてください．ビット4は8250Bの自己診断モード用です．ビット5-7は使用されていません（つねに0）．

また，モデム制御レジスタは入出力可能ですので，現在の状況を読み取ることができます．モデム制御レジスタは，シリアルI/Oポートのベースアドレス+4の位置にあります．

表4.9 モデム制御レジスタ

ビット	内容
7-5	つねに0
4	8250B/16550を自己診断モードにする
3	1 = 8250B/16550の割り込み要求信号を処理する
2	1 = Heyesコマンド内蔵モデムの電源オンリセット
1	1 = RS(RTS)信号をオンにする
0	1 = ER(DTR)信号をオンにする

したがって，RS(RTS)をオンにしたい場合には以下のようなコーディングを行います．

```
mov    dx, PortBase ; I/Oポートのベースアドレス
add    dx, 4        ; モデム制御レジスタのアドレス
in     al, dx       ; 現在のモデム制御状況の取得
or     al, 02h      ; ビット1をオンにします
out    dx, al       ; モデム制御レジスタを設定します
```

逆に，オフにしたい場合には以下のようなコーディングになります．

```
mov    dx, PortBase ;  I/Oポートのベースアドレス
add    dx, 4        ; モデム制御レジスタのアドレス
```

```
in      al, dx          ; 現在のモデム制御状況の取得
and     al, not 02h     ; ビット1をオフにします
out     dx, al          ; モデム制御レジスタを設定します
```

モデムステータスレジスタ

モデムステータスレジスタ（MSR）のビット4-7は，現在の入力信号線のレベルをモニタしています．ビット0-3は前回ステータスを読み取ってから，信号のレベルが変化した場合にオンになりますが，通常は使用することはありません．

また，モデムステータスレジスタは入力専用で，シリアルI/Oポートのベースアドレス+5の位置にあります．

表4.10　モデムステータスレジスタ

ビット	内　容
7	CD(DCD)のレベル
6	CI(RI)のレベル
5	DR(DSR)のレベル
4	CS(CTS)のレベル
3	CD(DCD)の変化
2	CI(RI)の変化
1	DR(DSR)の変化
0	CS(CTS)の変化

モデムステータスレジスタを読み込むには，以下のようなコーディングを行います．

```
mov     dx, PortBase ; I/Oポートのベースアドレス
add     dx, 5        ; モデムステータスレジスタのアドレス
in      al, dx       ; 現在のモデム制御状況の取得
```

リスト4.5に信号線関連の関数をのせておきます．

リスト4.5　信号線関連の関数

```
;*************************************************************************
;***                                                                   ***
;***     File Name       :  SIGCTL.ASM                                 ***
;***     Description     :  信号線の設定および状況の取得               ***
;***                                                                   ***
;*************************************************************************


include         std.inc
include         rsdrv.inc

                .code

;**************************************************************************
;*                                                                        *
;*      ER/RS信号の設定と状態の取得                                       *
;*      void        SetErOn(void);    ER 信号をオンにする                 *
;*      void        SetErOff(void);   ER 信号をオフにする                 *
;*      void        ChackEr(void);    ER 信号の状態の取得                 *
;*      void        SetRsOn(void);    RS 信号をオンにする                 *
;*      void        SetRsOff(void);   RS 信号をオフにする                 *
;*      void        ChackRs(void);    RS 信号の状態の取得                 *
;*                                                                        *
;**************************************************************************
SetErOn         proc    uses bx dx
                call    GetMcrStatus            ;モデム制御レジスタの読み取り
                or      al, DTR                 ; Bit 0をオンにします
                out     dx, al                  ;モデム制御レジスタの設定
                ret
SetErOn         endp

SetErOff        proc    uses bx dx
                call    GetMcrStatus            ;モデム制御レジスタの読み取り
                and     al, not DTR             ; Bit 0をオフにします
                out     dx, al                  ;モデム制御レジスタの設定
                ret
SetErOff        endp

CheckEr         proc    uses bx dx
                call    GetMcrStatus            ;モデム制御レジスタの読み取り
                xor      ax,ax
                .if bl & DTR                    ; Bit 0のチェック
                    inc      ax
                .endif
                ret
CheckEr         endp

SetRsOn         proc    uses bx dx
                call    GetMcrStatus            ;モデム制御レジスタの読み取り
                or      al, RTS                 ; Bit 1をオンにします
                out     dx, al                  ;モデム制御レジスタの設定
                ret
SetRsOn         endp

SetRsOff        proc    uses bx dx
                call    GetMcrStatus            ;モデム制御レジスタの読み取り
                and     al, not RTS             ; Bit 1をオフにします
                out     dx, al                  ;モデム 制御レジスタの設定
                ret
SetRsOff        endp

CheckRs         proc    uses bx dx
                call    GetMcrStatus            ;モデム制御レジスタの読み取り
                xor      ax,ax
                .if bl & RTS                    ; Bit 1のチェック
                    inc      ax
                .endif
                ret
CheckRs         endp

;**************************************************************************
;*                                                                        *
;*      モデム制御レジスタの現在の値の読み込み                            *
;*                                                                        *
;**************************************************************************
GetMcrStatus    proc    near private
```

```
        mov     dx, PortBase            ; I/Oポートのベースアドレス
                                        ;   PortBaseは rsdrv.inc 内で
                                        ;  外部定義されています
        add     dx, 4                   ;モデム制御レジスタのアドレス
        in      al, dx                  ;現在のモデム制御状況の取得
        mov     bl, al                  ;値の待避
        ret
GetMcrStatus    endp

;*************************************************************************
;*                                                                       *
;*      CD/CI/DR/CS信号の状態の読み込み                                  *
;*      void        ChackCd(void);    CD信号の状態の取得                 *
;*      void        ChackCi(void);    CI信号の状態の取得                 *
;*      void        ChackDr(void);    DR信号の状態の取得                 *
;*      void        ChackCs(void);    CS信号の状態の取得                 *
;*                                                                       *
;*************************************************************************
CheckCd         proc    uses bx dx
                call    GetMsrStatus    ;モデムステータスレジスタの読み取り
                xor     ax,ax
                .if bl & DCD            ; Bit 7のチェック
                    inc     ax
                .endif
                ret
CheckCd         endp

CheckCi         proc    uses bx dx
                call    GetMsrStatus    ;モデムステータスレジスタの読み取り
                xor     ax,ax
                .if bl & RI             ; Bit 6のチェック
                    inc     ax
                .endif
                ret
CheckCi         endp

CheckDr         proc    uses bx dx
                call    GetMsrStatus    ;モデムステータスレジスタの読み取り
                xor     ax,ax
                .if bl & DSR            ; Bit 5のチェック
                    inc     ax
                .endif
                ret
CheckDr         endp

CheckCs         proc    uses bx dx
                call    GetMsrStatus    ;モデムステータスレジスタの読み取り
                xor     ax,ax
                .if bl & CTS            ; Bit 4のチェック
                    inc     ax
                .endif
                ret
CheckCs         endp

;*************************************************************************
;*                                                                       *
;*          モデムステータスレジスタの現在の値の読み込み                 *
;*                                                                       *
;*************************************************************************
GetMsrStatus    proc    near private
                mov     dx, PortBase    ; I/Oポートのベースアドレス
                                        ;   PortBaseは rsdrv.inc 内で
                                        ;  外部定義されています
                add     dx, 5           ;モデムステータスレジスタのアドレス
                in      al, dx          ;現在のモデムステータスの取得
                mov     bl, al          ;値の待避
                ret
GetMsrStatus    endp

                end
```

4-4 送受信処理

ここでは，シリアル通信の主体となる送受信処理に関して解説します．

◎回線との送受信処理は，ハードウェア割り込みを使用して行います．

◎送信と受信が同じ割り込みを使用しますので，割り込みハンドラ内では割り込み識別レジスタを使用して割り込みの種類を確認します．

◎プログラムとのデータの受け渡しは，リングバッファを使用します．

◎送受信とも，RS/CSフロー制御およびXON/XOFFフロー制御に対応できるようにします．

◎FIFOモードにも対応できるようにします．

8250B系シリアルコントローラは，割り込み状態が発生した場合，内部的に優先順位づけを行いますので，受信割り込みと送信空き割り込みが同時に起きることはありません．そこで，ハードウェア割り込みの先頭で割り込み識別レジスタの内容を見て，割り込み処理を判定します（表4.11）．

このとき，下位3ビットだけを見て，04hのときは受信データレディ割り込み（FIFOモード時のタイムアウト割り込みも同様と考えて問題ありません），02hのときは送信レジスタ空き割り込みと考え，それ以外には対応しない（発生しないはずですが）ようにすれば問題ありません．

表4.11	割り込み識別レジスタ
ビット	内　　容
7,6	11 = FIFOバッファが使用可能
5,4	つねに0
3-0	割り込み原因識別
	0001 割り込み要求なし
	0110 ラインステータス割り込み
	0100 受信データレディ割り込み
	1100 タイムアウト割り込み（FIFOモードのみ）
	0010 送信レジスタ空き割り込み
	0000 モデムステータス割り込み

受信データレディ割り込み

まず，受信側から見ていきましょう．受信データレジスタ，受信FIFOバッファは，シリアルI/Oポートのベースアドレスの位置に入力専用レジスタとして位置しています．したがって，ここから受信データを読み出すことができます．最初に処理しなければいけないのは，XON/XOFF文字です．当然，XON/XOFFフロー制御を行っていない場合には無視すればよいのですが，フロー制御を行っている場合には，これらは，制御用の情報ですので，最初に処理しなければなりません．

XONを受信した場合には，前の状態が送信不可の場合には，送信可能フラグをオンにします，また，実際に送信すべきデータがある場合には，送信レジスタ空き割り込みを許可します．

XOFFを受信した場合には，前の状態が送信可能の場合には，送信可能フラグをオフにし，送信レジスタ空き割り込みを禁止します，

どちらでもない場合には通常の受信文字ですので，受信リングバッファに空きがあるかを確認して，空きがある場合には，受信文字を受信リングバッファに書き込んで，ポインタを進めます．また，受信リングバッファ中の受信文字数が限度（通常，受信リングバッファサイズの3/4程度）を越えた場合には，相手システムにXOFFを送信する（XON/XOFFフロー制御）かまたはCS信号を落とす（RS/CSフロー制御）ことにより，相手システムに受信不可だということを伝えます．

また，FIFOバッファ使用時は，1文字読み込んでもまだ受信バッファに文字データがある場合には，ラインステータスレジスタの受信データレディを見ることにより判定できますので，このビットが0になるまで，読み込みます（表4.12）

表4.12	ラインステータスレジスタ
ビット	内　　容
7	1 = FIFOバッファ中のデータにPE，FE，OEがあった
6	1 = 送信保持レジスタ・送信シフトレジスタが空 FIFOモード時は，送信FIFOレジスタ+シフトレジスタが空
5	1 = 送信保持レジスタが空 FIFOモード時は，送信FIFOレジスタが空
4	1 = ブレーク信号を検知したとき
3	1 = フレーミングエラー
2	1 = パリティエラー
1	1 = オーバーランエラー
0	1 = 受信データレディ標識

送信レジスタ空き割り込み

送信レジスタ空き割り込みを許可すると，送信保持レジスタまたは送信FIFOバッファが空いている状態に割り込みが発生します．単純に割り込みを許可してしまうと，送信しているとき以外はつねに割り込みが発生してしまうため，無駄な処理が多くなってしまいます．そこで，そういう状態にならないように，送信レジスタ空き割り込みの許可をこまめに制御するようにします．

すなわち，プログラムから送信要求がでた時点で，送信リングバッファに1文字追加し，送信レジスタ空き割り込みを許可します．シリアル通信割り込みハンドラでは，送信レジスタ空き割り込みの場合に，送信すべき文字があれば送信リングバッファから1文字を取り出して，送信保持レジスタまたは送信FIFOバッファに書き込んで，送信ポインタを1進めます．割り込み終了時点で，まだ送信すべき文字が残っている場合は次回の送信のために送信レジスタ空き割り込みを許可し，残っていない場合には送信レジスタ空き割り込みを禁止します．

送信保持レジスタ，送信FIFOバッファは，シリアルI/Oポートのベースアドレスの位置に出力専用レジスタとして位置しています．したがって，ここに送信データを書き込むことができます．

今回のサンプルプログラムでは対応していませんが，FIFOモードの場合には，送信レジスタ空き割り込みが発生した時点では，送信FIFOバッファは空の状態ですから，この時点で1〜16文字を書き込むことができます．さらに，送信割り込み処理期間中，受信データレディ割り込みも禁止しています．

また，とくに全二重通信が必要ない場合などは，受信処理だけを割り込み処理で駆動して，送信処理に関してはポーリングモードでコーディングすることができます．この場合にはラインステータスレジスタのビット5がオンであれば，送信保持レジスタまたは送信FIFOバッファが空いていますので，1文字（文字モード）〜16文字（FIFOモード）を書き込むことができます．

受信処理に関しては，よほど通信速度が低速でない限り，割り込み駆動にしないとオーバーランエラーが発生する可能性があります．

割り込み処理からの復帰

割り込みルーチンの最後の部分で，割り込みコントローラ8259Aに対して，EOIコマンドを発行しますが，スレーブに対する処理の場合には，EOIコマンドをスレーブの8259Aだけではなく，マスタの8259Aへも発行しなければなりません．

送受信処理のプログラムをリスト4.6にのせておきます．

リスト4.6

```
;****************************************************************************
;***                                                                     ***
;***        File Name          :  RSDRV.ASM                              ***
;***        Description        :  RS-232C Drive routine                  ***
;***                                                                     ***
;****************************************************************************


include         std.inc
include         rsdrv.inc

                .data

                extrn    PortBase:word            ; RS-232C 基本I/Oポートアドレス
                extrn    ImrPort:word             ; 8259 IMRの I/Oポートアドレス

XFlowFlag       byte     ON                       ; XON/XOFF制御フラグ
HFlowFlag       byte     ON                       ; RS/CS制御フラグ

SendEnable      word     ON                       ; OFF = 送信不可，ON = 送信可能
SendSize        word     0                        ;送信待ちの文字数
SendBuff        byte     SNDBUFSIZE dup (?)       ; 送信バッファ
SendRPtr        word     0                        ;送信バッファ読取ポインタ
SendWPtr        word     0                        ;送信バッファ書込ポインタ

RecvEnable      word     ON                       ; OFF = 受信不可，ON = 受信可能
RecvSize        word     0                        ;受信済みで未処理の文字数
RecvRPtr        word     0                        ;受信バッファ読取ポインタ
RecvWPtr        word     0                        ;受信バッファ書込ポインタ
RecvBuff        byte     RCVBUFSIZE dup (?)       ; 受信バッファ

                .code

;*****************************************************************************
;*                                                                           *
;*          RS-232Cハードウェア割り込みルーチン                              *
;*                                                                           *
;*****************************************************************************
rsHandler       proc     far uses ax bx dx ds
                mov      ax, @data
                mov      ds, ax
                assume   ds:@data

                mov      dx, PortBase                ; I/Oポートベースアドレス
                add      dx, IID                     ;割り込み識別レジスタ
                in       al, dx
                sub      dx, IID                     ;元に戻す
                and      al, 07h                     ;下位3ビットのみ処理する

                .if al == 04h                        ;受信割り込み/タイムアウトのとき
                    .repeat
                        in      al, dx                   ;受信データを読み込む
                        mov     ah, XFlowFlag            ; XON/XOFF制御フラグ
                        .if ax == (XON or 100h)          ; XON/XOFFフローでXONを受信
                                                         ;送信可能状態になった
                            .if SendEnable == OFF        ;前の状態が送信不可状態のとき
                                mov     SendEnable, ON   ;フラグを送信可能に
                                .if SendSize != 0        ;送信するものがあるとき
                                    inc     dx           ;割り込み許可レジスタ
                                    in      al, dx       ;現在の状況の読み取り
                                    or      al, TxEE     ;送信割り込み可能
                                    out     dx, al       ;
                                    dec     dx           ;元に戻す
                                .endif
                            .endif
                        .elseif ax == (XOFF or 100h)     ; XON/XOFFフローでXOFFを受信
                                                         ;送信不可の状態になった
                            .if SendEnable == ON         ;前の状態が送信可能状態のとき
                                mov     SendEnable, OFF  ;フラグを送信不可に
                                inc     dx               ;割り込み許可レジスタ
                                in      al, dx           ;現在の状況の読み取り
                                and     al, (not TxEE)   ;送信割り込み禁止
                                out     dx, al           ;
                                dec     dx               ;元に戻す
                            .endif
                        .else                            ;通常文字の受信
                            .if RecvSize != RCVBUFSIZE   ;受信バッファに空きがある
                                mov     bx, RecvWPtr     ;受信バッファ書込ポインタ
                                mov     RecvBuff[bx], al ;受信バッファに文字を格納
                                inc     bx               ;書込ポインタを進める
                                and     bx, RCVBUFSIZE-1 ;バッファのラップラウンド
                                mov     RecvWPtr, bx     ;ポインタを格納する
                                inc     RecvSize         ;受信済みで未処理の文字数
                                .if RecvSize > RECVFULL  ;受信文字限界を越えた
                                    call    SendXoffCmd  ;相手に受信不可を伝える
                                .endif
                            .else
                                call    SendXoffCmd      ;相手に受信不可を伝える
                            .endif
                        .endif

                        add     dx, LSR                  ;ラインステータスレジスタ
                        in      al, dx                   ;現在の値
                        sub     dx, LSR                  ;元に戻す
                    .until !(al & 01h)                   ;受信データがなくなるまで

                .elseif al == 02h                        ;送信割り込みが発生
                    inc     dx                           ;割り込み許可レジスタ
                    in      al, dx
                    and     al, (not (TxEE or RxRE))     ;送受信割り込み禁止
                    out     dx, al
                    dec     dx                               ;元に戻す

                    mov     ah, RxRE                     ;割り込み許可のためのワーク
                                                         ;受信割り込み可能
                    .if SendSize != 0                    ;送信待ちがある場合
                        mov     bx, SendRPtr             ;送信バッファ読取ポインタ
                        mov     al, SendBuff[bx          ; 1文字を取り出す
                        out     dx, al                   ; 1文字を送信する
                        inc     bx                       ;ポインタを進める
                        and     bx, SNDBUFSIZE-1         ;バッファのラップラウンド
                        mov     SendRPtr, bx             ;ポインタを格納する
                        dec     SendSize                 ;送信待ち文字数を減らす
                        .if !zero?                       ;まだ文字が残っている
                            or      ah, TxEE             ;送信割り込み可能
                        .endif
                    .endif
                    inc     dx                           ;割り込み許可レジスタ
                    in      al, dx                       ;現在の許可状況の取得
                    or      al, ah                       ;送受信割り込み許可？
                    out     dx, al                       ;
                    dec     dx                           ;元に戻す
                .endif

                mov     al, EOI                          ; EOIコマンド
                .if ImrPort == 0A1h                      ; IMRのポートアドレス
                    out     PIC2, al                     ;割り込みコントローラにEOIを出力
                .endif
                out     PIC, al                          ;割り込みコントローラにEOIを出力

                assume  ds:nothing
                iret
rsHandler       endp


;*****************************************************************************
;*                                                                           *
;*        相手システムに受信が出来ないことを伝える（内部関数）               *
;*                                                                           *
;*            XON/XOFF制御の場合は，XOFFを送信する                           *
;*            RS/CS制御の場合は，RSをオフにする                              *
;*                                                                           *
;*****************************************************************************
SendXoffCmd     proc    near private
                assume  ds:@data
                .if RecvEnable == ON                     ;前の状態が受信可能の場合
                    mov     RecvEnable, OFF              ;受信を受信不可に
                    .if XFlowFlag != OFF                 ; XON/XOFF制御を行うとき
                        add     dx, LSR                  ;回線ステータスレジスタ
                        .repeat
                            in      al, dx
                        .until (al & TxRDY)              ;送信ホールディングレジスタ
                                                         ;が空になるまで
                        sub     dx, LSR                  ;元に戻す
                        mov     al, XOFF
                        out     dx, al                   ; XOFFを送信する
                    .endif
                    .if HFlowFlag == ON                  ; RS/CSフロー制御を行うとき
                        add     dx, MCR                  ;モデム制御レジスタ
                        in      al, dx                   ;現在の状況の読み取り
                        and     al, not RTS              ;
                        out     dx, al                   ; RS(RTS)をオフにする
                        add     dx, MCR                  ;元に戻す
                    .endif
                .endif
                assume  ds:nothing
                ret
SendXoffCmd     endp


;*****************************************************************************
;*                                                                           *
;*        1文字受信ルーチン                                                  *
;*        int      rsGetch(void);                                            *
;*        戻り値： 0以上   受信した文字                                      *
;*                 -1      受信した文字はない                                *
;*                                                                           *
;*****************************************************************************
rsGetch         proc    uses bx dx
                assume  ds:@data
                .if RecvSize != 0                        ;受信済みの未処理文字がある
                    cli
                    mov     dx, PortBase                 ; I/Oポートのベースアドレス
                    mov     bx, RecvRPtr                 ;受信バッファ読取ポインタ
                    xor     ah, ah                       ;上位バイトをクリア
                    mov     al, RecvBuff[bx              ; 1文字読み取る
                    inc     bx                           ;ポインタを進める
                    and     bx, RCVBUFSIZE-1             ;バッファのラップラウンド
                    mov     RecvRPtr, bx                 ;ポインタを戻す
                    .if ax == 0                          ;受信文字がNULLのときは無視
                        mov     ax, -1
                    .endif
                    dec     RecvSize                     ;未処理文字数を減らす
                    .if RecvEnable != ON && ¥            ;受信不可状態で
                        RecvSize <= RECVEMPTY            ;受信バッファが限界以下
                        mov     RecvEnable, ON           ;フラグを受信可能に
                        .if XFlowFlag == ON              ; XON/XOFF制御をしている場合
                            add     dx, LSR              ;回線ステータスレジスタ
                            .repeat
                                in      al, dx           ;ステータスの読み取り
```

```
            .until al & TxRDY       ;送信可能になるまで待つ
            sub     dx, LSR         ;元に戻す
            mov     al, XON         ;
            out     dx, al          ; XONを送信する
        .endif
        .if HFlowFlag == ON         ; RS/CSフロー制御を行うとき
            add     dx, MCR         ;モデム制御レジスタ
            in      al, dx          ;現在の状況の読み取り
            or      al, RTS         ;
            out     dx, al          ; RS(RTS)をオンにする
            sub     dx, MCR         ;元に戻す
        .endif
    .endif
    .else
        mov     ax, -1              ;受信済み文字がない場合
    .endif
    sti
    assume  ds:nothing
    ret
rsGetch     endp

;*****************************************************************************
;*                                                                           *
;*          1文字送信ルーチン                                                *
;*          void    rsPutch(char SendChar);                                  *
;*          パラメータ: SendChar    送信文字                                 *
;*          戻り値:     なし                                                 *
;*                                                                           *
;*****************************************************************************
rsPutch         proc    uses bx dx, SendChar:byte
                assume  ds:@data
                mov     al, SendChar            ;送信すべき文字
                .if SendSize != SNDBUFSIZE      ;送信バッファがフルでない
                    mov     bx, SendWPtr        ;送信バッファ書込ポインタ
                    mov     SendBuff[bx], al    ;バッファに格納する
                    inc     bx                  ;ポインタを進める
                    and     bx, SNDBUFSIZE-1    ;バッファのラップラウンド
                    mov     SendWPtr, bx        ;ポインタを格納する
                    inc     SendSize            ;送信待ち文字数を増やす
                    .if SendEnable == ON        ;送信可能状態なら
                        mov     dx, PortBase    ; I/Oポートのベースアドレス
                        inc     dx              ;割り込み許可レジスタ
                        in      al, dx          ;現在の状況の読み取り
                        or      al, TxEE        ;送信割り込み可能
                        out     dx, al          ;
                    .endif
                .endif
                ret
rsPutch         endp


                end
```

4-5 ブレーク信号の送出

シリアル通信では，通常のデータ転送以外に，送受信の中断の目的でブレーク信号と呼ばれる特別な信号を送ることができます．ここでは，ブレーク信号の送出方法と，相手システムからのブレーク信号の検知方法に関して解説します．

◎ブレーク信号とはシリアル出力がスペース状態になることです．

◎ブレーク信号は相手システムが検知できるように，一定時間その状態を保っておかなければなりません．最近のパソコン通信などでは，回線速度も最低1200bpsですので，150ミリ秒程度保持しておけば十分です．

ブレーク信号を送出するためには，ライン制御レジスタのビット6（Set Break）をオンにします．このビットをオン（1）にセットすると，シリアル出力が強制的にスペース状態にされ，このビットがオフ（0）になるまでその他の送信活動とは無関係にその状態が保持されます．

ライン制御レジスタは入出力可能ですので，現在の状況を読み取ることも可能で，シリアルI/Oポートのベースアドレス+3の位置にあります．

したがって，ビット6をオンにしたい場合には，以下のようなコーディングを行います．

```
mov     dx, PortBase ; I/Oポートのベースアドレス
add     dx, 3        ; ライン制御レジスタのアドレス
in      al, dx       ; 現在の回線制御状況の取得
or      al, 40h      ; ビット6をオンにします
out     dx, al       ; ライン制御レジスタを設定します
```

このブレーク信号は，相手システムが検知できるだけの十分な時間，その状態を保持し

ます．前述のように，よほど低速の端末を接続しないかぎり，150ミリ秒程度ブレーク状態を保持しておけば十分です．

この時間計測に関しては，システムBIOSのなかの待ち時間の経過待ちを使用する方法とシステムタイマを使用する方法の2種類があります．両者に関しては「第6章　タイマ編」で詳細に解説しますが，概要だけまとめておきます．

システムBIOSのなかの待ち時間の経過待ちは，cx:dxレジスタでマイクロ秒単位で待ち時間を指定することができます．ただし，実際には1/1024秒単位で更新されますので，約976マイクロ秒単位にカウントアップされます．システムタイマは，1/18.2秒に1回発生するタイマ割り込みを使用してカウントアップします．すなわち，約55ミリ秒に1ずつカウントアップされますので，4回待てば，最悪でも165ミリ秒は待つことになりますので，十分な値といえるでしょう．サンプルプログラムでは，前者のシステムBIOSを使用しています．また，システムタイマを使用する場合に関しては，「第6章　タイマ編」で解説する，TimerWait関数を使ってコメントとして入れています．

ブレーク信号の送出を終了するにはライン制御レジスタのビット6をオフします．以下にコーディング例をのせておきます．

```
mov     dx, PortBase ; I/Oポートのベースアドレス
add     dx, 3        ; ライン制御レジスタのアドレス
in      al, dx       ; 現在のライン制御状況の取得
and     al, not 40h  ; ビット6をオフにします
out     dx, al       ; ライン制御レジスタを設定します
```

これらをまとめたのが，リスト4.7のプログラムです．

つぎに，ブレーク信号の検知方法ですが，これはラインステータスレジスタのビット4（Break Interrupt）を監視すれば，判定することができます．このビットがオンのときは，受信データ入力のスペース状態が1ワード（スタートビットからストップビットまで）の送信期間を越えて継続していることを示しています．このビットはCPUがラインステータスレジスタの内容を読み取ると0にセットされます．また，ブレーク信号を検出した場

リスト4.7　　**ブレーク信号の送出**

```
;*****************************************************************************
;***                                                                       ***
;***        File Name         :  BREAK.ASM                                 ***
;***        Description       :  ブレーク信号の送出                         ***
;***                                                                       ***
;*****************************************************************************


include         std.inc
include         rsdrv.inc

                .data
                extrn   PortBase:word           ; RS-232C基本I/Oポートアドレス

                .code
;               extrn   TimerWait:near

;******************************************************************************
;*                                                                            *
;*          ブレーク信号の送出                                                *
;*          void            rsSendBreak(void);                                *
;*                                                                            *
;******************************************************************************
rsSendBreak     proc    uses cx dx
                assume ds:@data
                mov     dx, PortBase            ; I/Oポートのベースアドレス
                add     dx, LCR                 ;ライン制御レジスタのアドレス
                in      al, dx                  ;現在のライン制御状況の取得
                or      al, 40h                 ; Bit 6をオンにします
                out     dx, al                  ;ライン制御レジスタを設定します

                push    ax

;*** システムBIOS を使用した方法 ***
                mov     cx, 2                   ; 2*65536 + 18928 = 150000
                mov     dx, 18928               ; cx:dx = 150000μs
                INT15   86h                     ;待ち時間の経過待ち

;*** タイマBIOS を使用した関数を呼びだす方法 ***
;               mov     ax,4
;               push    ax
;               call    TimerWait               ; (3~4)*55ms だけ待つ
;               pop     ax

                pop     ax

                and     al, not 40h             ; Bit 6をオフにします
                out     dx, al                  ;ライン制御レジスタを設定します
                ret
rsSendBreak     endp

                end
```

合，受信レジスタに1つだけ00hがロードされます．

このビット4はラインステータスレジスタのエラー条件となっていて，受信回線ステータス割り込みが許可されている場合には，割り込みが発生します．

ただし，ブレーク信号検出は，通常はホスト側が行うものであり，一般的な通信端末プログラムでは対応していない場合があります．今回のサンプルプログラムでも，ブレーク信号の検知は行っていません．

第5章

98からDOS/Vの世界へ

マウス編

ここでは，周辺機器のひとつとして，マウスを用いたプログラミングテクニックを解説します．PS/2のポインティングデバイスコントローラ用ROM BIOSを使用する方法と，AT標準のシリアルインターフェースを使用する方法の2種類がありますが，本書では後者の方法について取り上げます．

5-1 基礎知識

マウスを用いたプログラミング方法は，DOS/Vアプリケーションとしては2種類考えておく必要があります．

ひとつは，PS/2互換機，ATバス用のPS/2マウス対応インターフェースボードのついているAT互換機，または最近のDynaBookなどを対象にして専用のプログラム開発を行うものです．これはPS/2の既存のポインティングデバイスコントローラ用ROM BIOSを使用しますので，わざわざドライバなどをインストールする必要がないというメリットがありますが，その反面ポインティングデバイスコントローラ用ROM BIOSをもたないAT互換機では使用できないというデメリットがあります（某有名ワープロのDOS/V版も，こちらのほうを採用しているので，AT互換機上ではマウスが使えません）．

もうひとつは，AT標準ともいえるシリアルマウスインターフェースです．これはIBM DOSバージョンJ5.0x/Vに付属しているMOUSE.COMが別途必要となりますが，前述のPS/2互換機種などに加え，すべてのAT互換機で使用できるというメリットがあります．

本書では，後者のシリアルマウス用インターフェースを用いたマウスプログラミングについて，詳細にプログラムサンプルを交えながら解説していくことにしましょう．

マウスインターフェースを使用可能にする

マウスインターフェースを使用可能とするためには，前述したとおりDOS付属のMOUSE.COMが必要になります．MOUSE.COMはDOSプロンプトが表示されている状態であればいつでも起動できます．いつも使用するものであれば，AUTOEXEC.BATなどに以下のような1行を入れておくとよいでしょう．

```
C:¥DOS¥MOUSE.COM
```

または

```
LH C:¥DOS¥MOUSE.COM
```

マウスドライバがメモリに常駐すると，図5.1のようなメッセージが表示されます．これでマウスインターフェースが使用可能となります．

図5.1

マウスドライバ組み込みのメッセージ

```
Microsoft (R) Mouse Driver Version 7.04
Copyright (C) Microsoft Corp. 1983-1990.  All rights reserved.
Copyright (C) IBM Corp. 1991-1992.  All rights reserved.
Mouse driver installed
```

マウスインターフェースの使用上の考慮点

PC-9801シリーズでは，テキストとグラフィックの混在表示が可能なのは有名な話しですが，IBM PC/AT，PS/2などではテキストモードではテキストしか使用できません．このためマウスインターフェースにはグラフィックモードでのみ使用できるファンクションや，テキストモードでのみ使用できるファンクションがでてしまいました．また，マウスカーソルの座標系は，テキストモードでもグラフィックモードでもグラフィック座標を採用しているため，画面モードによって異なる解像度が発生します．また，テキストモードでマウスを使おうと思うと，実際のテキスト座標への変換作業も必要となります．

こういったことも踏まえて，サンプルプログラムでは，画面モードを判断してテキストモードはテキスト座標値（表示可能桁数，行数）で，グラフィックモードはグラフィック座標値で制御できるように作られています．

マウスインターフェースの使用上の予備知識

座標

座標は画面の位置を表す単位で，左上端を原点0として，右方向，下方向に向かって昇順にカウントされる値です（図5.2）．

グラフィック座標は，0からそれぞれの画面モードの解像度と等しく，VGAモード（画面モード72h）であれば，Xは0～639，Yは0～479となります．

サンプルプログラムでは，この値は1からの相対になるようになっています．また，xn,ynについては画面モードによっても変わりますので，表2.2のビデオモード番号（P.28）を参照してください（表5.1）．

テキスト座標は，0からそれぞれの画面モードで表示できる文字の桁数，行数と等しくなります．通常のVGAで80×25行モードの場合は，Xが0～79，Yが0～24となります．

サンプルプログラムでは，1からの相対でXは1～xn，Yは1～ynになるようになっています．xn,ynについては画面モードによって変わりますので，リスト2.1のGetVideoInfo()関数（P.31）の使用説明を参照してください．

テキストモード，グラフィックモードでの座標系

(0,0)

(xn,yn)

図5.2

画面と座標

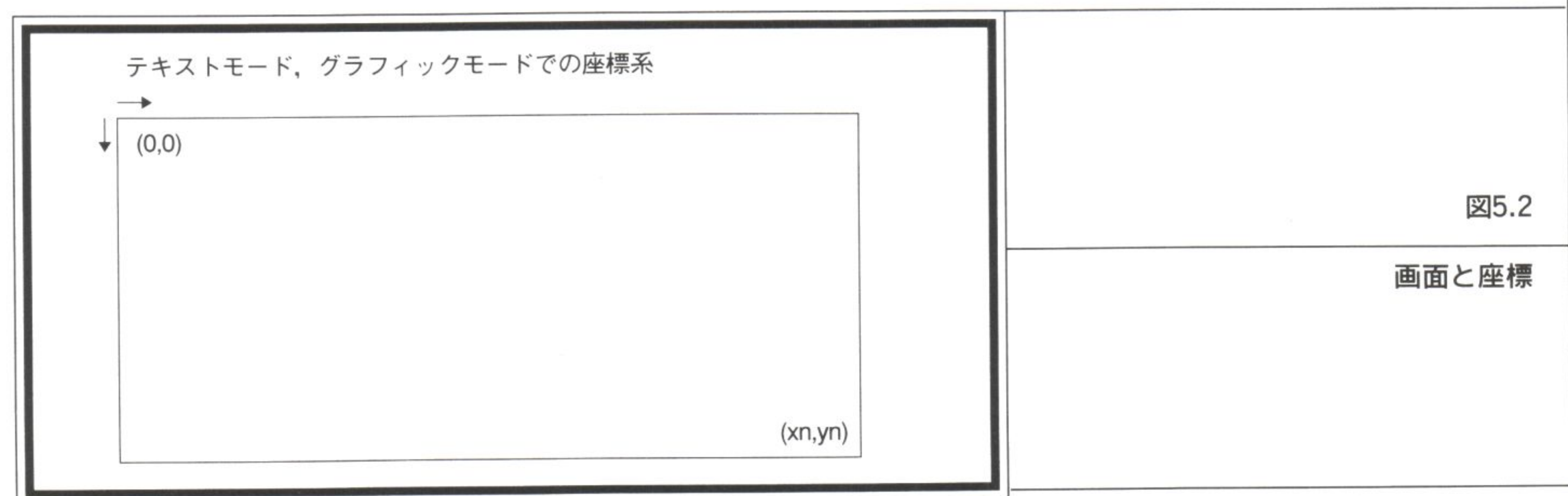



表5.1　マウスドライバがサポートする画面モード

画面モード	カラー	モード	仮想画面	日本語モード	英語モード
0	16	テキスト	640×200		○
1	16	テキスト	640×200		○
2	16	テキスト	640×200		○
3	16	テキスト	640×200	○	○
4	4	グラフィック	640×200		○
5	4	グラフィック	640×200		○
6	2	グラフィック	640×200		○
7	単色	テキスト	640×200		○
D	16	グラフィック	640×200		○
E	16	グラフィック	640×200		○
F	単色	グラフィック	640×350		○
10	16	グラフィック	640×480		○
11	2	グラフィック	640×480	○	○
12	16	グラフィック	640×480	○	○
13	256	グラフィック	640×200		○
72	16	グラフィック	640×480	○	
73	16	テキスト	640×480	○	
83	16	V-Text		○	

マウスカーソル

DOS/Vでのマウスカーソルはテキストモード，グラフィックモードの2種類が用意されています．カーソルは表示パターンでAND演算を行い（表示する形にクリアする），次にXOR演算（表示する）という2段階の処理を行うことで表示を行っています．そのため，グラフィックカーソルもテキストカーソルも2種類のカーソルパターン情報が必要になります．

(1)　グラフィックカーソル

グラフィックカーソルは画面モードがグラフィックモードのときに表示されます．カーソルは16×16ドットからなっており，ANDマスク，XORマスクの2種類を用意する必要があります．ANDマスクはカーソル形状を表示するために，カーソルの形に表示部分をクリアするためのパターンです．基本的には表示したい形はビット0（OFF）で表現します．XORマスクはカーソルを実際に表示するためのパターンです．表示したい形はビット1（ON）で表します．

```
AND マスクパターン      XOR マスクパターン      ・AND->XOR->結果パターン

001111111111111111    000000000000000000    00________________
000111111111111111    010000000000000000    010_______________
000011111111111111    011000000000000000    0110______________
000001111111111111    011100000000000000    01110_____________
000000111111111111    011110000000000000    011110____________
000000011111111111    011111000000000000    0111110___________
000000001111111111    011111100000000000    01111110__________
000000000111111111    011111110000000000    011111110_________
000000000011111111    011111111000000000    0111111110________
000000000001111111    011111111100000000    01111111110_______
000000000000111111    011111111110000000    011111111110______
000100001111111111    011001100000000000    01100110__________
001100001111111111    010001100000000000    010_0110__________
111110000111111111    000000110000000000    00__ 0110_________
111110000111111111    000000110000000000    _____0110_________
111111000111111111    000000000000000000    ______000_________
```

上図のとおり，ANDマスクは0の部分をクリアするためのビットパターンです．上図では0の部分がカーソルの形になっています．また，XORマスクはカーソルの縁を残して，ほかはすべてビット1からのデータになっています．

図5.3　グラフィックカーソルのマスクパターン

グラフィックカーソルは画面の1点を指示するためのものです．そこで，前述のパターンにも指示点と呼ばれる1点が必要になります．図5.3の矢印カーソルの場合の指示点は(0,0)つまり，左上端が指示点となります．

(2)　テキストカーソル

テキストカーソルは画面モードがテキストモードのときに表示されます．テキストカーソルにはソフトウェアテキストカーソルと，ハードウェアテキストカーソルがありますが，後者のハードウェアテキストカーソルはDOS/V日本語モードではサポートされていませんので，本章での説明は割愛します．

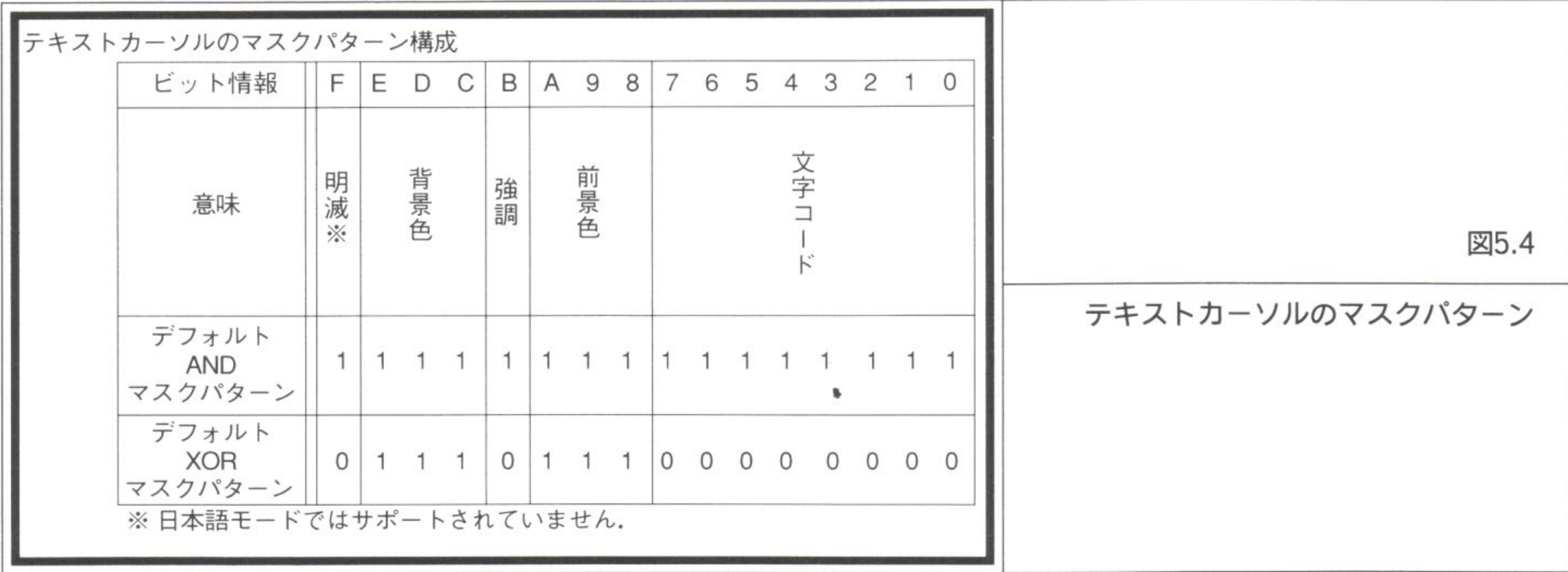
テキストカーソルにもANDマスクとXORマスクを使用しますが，ANDマスクとXORマスクはそれぞれ16ビットからなる図5.4の形式の情報です．

テキストカーソルのマスクパターン構成

ビット情報	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
意味	明滅※	背景色			強調	前景色			文字コード							
デフォルト AND マスクパターン	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
デフォルト XOR マスクパターン	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0

※ 日本語モードではサポートされていません．

図5.4

テキストカーソルのマスクパターン

下位7ビットのマスクパターンはあくまでも画面に表示されている文字に対するビットマスクです．

ボタン

マウスには通常2つのボタンがついています．ボタンは左右に並んでついており，これらには「押されている状態」と「離されている状態」という2つの状態があります．マウスを利用するプログラムでは，ボタンの各種状態とマウスの移動を組み合わせてマウス処理を実現します．

マウスの移動距離

マウスには移動面にボールがついており，マウスを移動するとこのボールが転がって，そのボールの動きからマウスの移動方向，距離などがわかるようなしくみになっています．この移動した距離の，どれだけ動いたかという数の単位は「マウスのΔ」と呼ばれ（数学ではこの量を「移動量」と呼び，移動量はΔまたはδで表されます），約0.5mmを1としたものになっています．ここで気をつけてほしいのが，この「マウスのΔ」の単位1が必ずしも画面の1ドットに相当するわけではないということです．実際のマウスの移動距離と，カーソルの移動距離は，ドライバAPIによって自由に設定することが可能で，この2つの値の割合を「移動比率」と呼び，x方向，y方向をそれぞれ設定することが可能になっています．

5.2 マウスインターフェースを使う

それでは，実際のマウスドライバAPIでどんなことができるのでしょうか？　ここではマウスAPIについて，どんなものがあるのか見ていくことにしましょう．表5.2の「参照サンプル関数名」は，サンプルとして作成したMOUSE.ASM（MS-C用関数）の，Cから呼び出すための関数名です．そのあと，マウスドライバAPIの説明に入ることにしましょう．

マウスインターフェースの呼び出し方法

マウスドライバAPIを呼び出す手順はいたって簡単で，以下のように呼び出しを行います．

```
mov     ax, 機能番号
int     33h
```


表5.2　マウスドライバ機能の一覧

マウスAPI機能名	機能NO	参照サンプル関数名
マウス機能の初期化	0	GetMouseState()
カーソルの表示	1	CtrlMouseCursor()
カーソルの消去	2	CtrlMouseCursor()
カーソル位置とボタン状態の読み取り	3	GetMouseCursorInfo()
カーソル位置のセット	4	SetMouseCursorPos()
ボタンを押した回数と最終位置の読み取り	5	GetMouseClickCount()
ボタンを離した回数と最終位置の読み取り	6	GetMouseReleaseCount
カーソル移動範囲の設定（X方向）	7	SetMousePhysBlock()
カーソル移動範囲の設定（Y方向）	8	SetMousePhysBlock()
グラフィックカーソルの形状設定	9	SetMouseCursorShape()
テキストカーソルのマスク設定	10	SetMouseCursorMode()
マウスの移動距離（マウスΔ）の読み取り	11	GetMouseMoveDis()
ユーザー割り込みの設定	12	SetMouseUserInt()
ライトペンエミュレーション機能開始	13	——
ライトペンエミュレーション機能終了	14	——
マウスの移動比率の設定	15	SetMouseDistance()
カーソル非表示域の設定	16	SetMouseClearRange()
倍速境界値の設定	19	SetMouseOverDrive()
ユーザー割り込みルーチンの差し替え	20	ResetMouseUserInt()
ドライバ状態保管用バッファサイズの取得	21	SaveMouseInfo()
ドライバ状態の保管	22	SaveMouseInfo()
ドライバ状態の回復	23	RestoreMouseInfo()
代替ユーザー割り込みルーチンの設定	24	SetExtMouseUserInt()
代替ユーザー割り込みルーチンのアドレス取得	25	GetExtUserIntAdr()
マウス感度の設定	26	SetMouseCoefficient()
マウス感度の取得	27	GetMouseCoefficient()
カーソル表示ページの設定	29	SetMouseCrtPage()
カーソル表示ページの取得	30	GetMouseCrtPage()
マウスドライバの使用禁止設定	31	ReleaseMouseDriver()
マウスドライバの使用禁止解除	32	ResetMouseDriver()
マウスドライバのリセット	33	SoftResetMouse()

また，サンプルプログラムでは以下のようなマクロを使用して，ファンクションの呼び出しを行っています．マクロについては，MOUSE.ASMの冒頭，STD.INCを参照してください．

MOUSE 0, 機能NO

そのほか，BX,CX,DXなど機能によって必要なレジスタもありますので，後述のファンクションの詳細説明を参照してください．

サンプルプログラムについて

本章で取り上げているマウス制御ルーチン（MOUSE.ASM）は，このままマウス制御ライブラリとして使えるように考慮して作成されています．このマウス制御ライブラリは，グラフィックモード用，テキストモード用を併用できるようになっています．このマウス制御ライブラリには以下のような特徴があります．

◎テキストモードでは1からの相対の桁数，行数で座標指定が可能になっています．

◎グラフィックモードでは1からの相対の，解像度に合ったグラフィック座標値が指定できます．

◎ユーザー割り込み（機能番号12，24）は，独自の割り込みルーチンをもち，Cからでも簡単に呼び出し，実行ができるようになっています．

リスト5.1 MOUSE.H

```
/*------------------------------------------------------------------------*/
/*                         マウス制御用ヘッダ                              */
/*                                               Author : T.Shibazaki     */
/*------------------------------------------------------------------------*/
#if !defined(MOUSE_H)
#define MOUSE_H

/*------------------------------------------------------------------------*/
/*  マウスカーソル制御定義                                                  */
/*                                                                        */
/*      マウスカーソルの消去，表示を制御するための定義です．                   */
/*    消去する場合は MC_OFF を，表示する場合は MC_ON をそれぞれ              */
/*    使用します．                                                          */
/*------------------------------------------------------------------------*/
#define     MC_ON       (0)
#define     MC_OFF      (1)

//------------------------------------------------------------------------
//  マウスボタン押下，離上制御定義
//
//      マウスのボタンが押された回数，離された回数を調べる関数用の
//    定義です．マウスボタンの左右どちらを使用するか指定する場合に
//    以下の定義を使用してください．
//------------------------------------------------------------------------
#define     MC_LEFTB    (0)
#define     MC_RIGHTB   (1)

//------------------------------------------------------------------------
//  テキストカーソル設定用定義
//
//      以下のフラグは，テキストカーソルのモード設定用に使用します．
//    日本語モードではソフトウェアカーソルのみが使用可能です．
//------------------------------------------------------------------------
#define     MCC_SOFT_CURSOR     (0)
#define     MCC_HARD_CURSOR     (1)

//------------------------------------------------------------------------
//  ユーザー割り込み用定義変数
//
//      以下のフラグは，ユーザー割り込みトリガ用，また割り込み関数
//    へのトリガ検知コードとしても用います．以下の定義をトリガとし
//    て使用する場合は，OR条件で接続して複数指定することも可能で．
//
//------------------------------------------------------------------------
#define     MCU_MOVE_CURSOR (0x0001U)       // カーソルが移動した
#define     MCU_DOWN_LEFTB  (0x0002U)       // 左ボタンが押された
#define     MCU_RELS_LEFTB  (0x0004U)       // 左ボタンが離された
#define     MCU_DOWN_RIGHTB (0x0008U)       // 右ボタンが押された
#define     MCU_RELS_RIGHTB (0x0010U)       // 右ボタンが離された
#define     MCUE_SHIFT_KEY  (0x0020U)       // Shiftキーと同時
#define     MCUE_CTRL_KEY   (0x0040U)       // Ctrlキーと同時
#define     MCUE_ALT_KEY    (0x0080U)       // Altキーと同時

//------------------------------------------------------------------------
//  マウス制御用構造体
//
//      座標値はビデオモードがテキストに設定されている場合は，テキスト
//    座標を，グラフィックモードに設定されている場合は，グラフィック座
//    標を，それぞれ返します．座標値はどちらも1からの相対です．
//
//      ボタン押下フラグは押されている状態のときは1，離されている状態
//    のときは0をセットします．
//------------------------------------------------------------------------
typedef struct  MOUSE_CURSOR_INFORMATION{
            int     CurX ;                  // X座標
            int     CurY ;                  // Y座標
            char    LeftButton ;            //左ボタン押下フラグ
            char    RightButton ;           //右ボタン押下フラグ
}M_CURSOR_INFO ;

#define ReleaseMouseUserInt()       GetMouseState()

//  マウス制御関数
extern  unsigned    GetMouseState(void) ;
extern  void        CtrlMouseCursor(int  sw) ;
extern  void        GetMouseCursorInfo(M_CURSOR_INFO  *mcinf) ;
extern  void        SetMouseCursorPos(int  x,int  y) ;
extern  int         GetMouseClickCount(int  btn,M_CURSOR_INFO *mcinf) ;
extern  int         GetMouseReleaseCount(int  btn,M_CURSOR_INFO *mcinf) ;
extern  void        SetMousePhysBlock(int x1,int y1,int x2,int y2) ;
extern  void        GetMouseMoveDis(int  *xdis,int  *ydis) ;
extern  void        SetMouseUserInt(unsigned flag,int  (*func)()) ;
extern  void        SetMouseDistance(int  xdis,int  ydis) ;
extern  void        SetMouseClearRange(int x1,int y1,int x2,int y2) ;
extern  void        SetMouseOverDrive(int  OVcnt) ;
extern  void        ResetMouseUserInt(unsigned flag,int  (*func)()) ;
extern  int         SaveMouseInfo(void) ;
extern  int         RestoreMouseInfo(void) ;
extern  void        SetExtMouseUserInt(unsigned flag,int  (*func)()) ;
extern  void        GetExtUserIntAdr(unsigned flag) ;
extern  void        SetMouseCoefficient(int xcoef,int ycoef,int ovcoef) ;
extern  void        GetMouseCoefficient(int *xcoef,int *ycoef,int *ovcoef) ;
extern  void        SetMouseCrtPage(int  crt_page) ;
extern  int         GetMouseCrtPage(void) ;
extern  int         ReleaseMouseDriver(void) ;
extern  void        ResetMouseDriver(void) ;
extern  int         SoftResetMouse(void) ;
extern  void        SetMouseCursorShape(int xpoint,int ypoint,char *ship) ;

#endif

/*-------------------------- End of File ---------------------------------*/
```

本文中サンプルとして取り上げているCプログラムの参照用として，マウス制御関数用ヘッダMOUSE.Hをリスト5.1に掲載します．本文中に掲載したマウス制御ルーチンはすべてテキストモード，およびグラフィックモードで使用できますが，本文中のCのサンプルはすべてテキストモードを対象として作成してあります．

マウスインターフェースAPI

マウス機能の初期化（機能番号：0）

このファンクションはマウスドライバがインストールされているか，マウスが接続されているかなどをチェックし，現在マウスが使用できる状態かどうかを返してくれます．また，マウスが使用可能な場合は，マウスドライバで使用する内部変数などを初期化します．初期化される変数は表5.3のとおりです．

ax = 0
int　33h
[戻り値]
ax = 0　マウスが使用できない
ax = −1　マウスが使用可能
bl = 　マウスボタンの数

マウスドライバAPIを使用するにあたって，この機能は必ず実行しなければなりません．リスト5.2に，当ファンクションの呼び出しルーチンと，そのルーチンの使用例を記します．

表5.3　初期化される内部変数

項目	状態
カーソル表示	0FF
カーソル位置	画面中央
テキストカーソル形状	反転表示
グラフィックカーソル形状	矢印カーソル
カーソル指示点	左上端（0，0）
カーソルのX方向移動可能範囲	0〜639
カーソルのY方向移動可能範囲	画面モードによる
マウスのX方向移動比率	マウスΔ8に対して，仮想画面8ドット
マウスのY方向移動比率	マウスΔ16に対して，仮想画面8ドット
ユーザー割り込み用マスク	すべて0FF

リスト5.2　マウス機能の初期化

【Cからの呼び出し】

```
{
    struct  MI_RET{
            unsigned    st : 8 ;                //マウス動作状態
            unsigned    bc : 8 ;                //ボタンの数
    }minit ;

    *((unsigned *)&minit) = GetMouseState() ;   // マウス使用不可チェック

    if ( !minit.st ) {                          //マウス使用出来ない？
        printf("MOUSE Driver not installed !!¥7¥n") ;
        return -1 ;
    }
}
```

【MASM6ルーチン】

```
;=====================================================================
;       Description : マウスの初期設定
;       Sequence    : unsigned  GetMouseState(void) ;
;       戻り値      : struct {
;                           usnigned State :8 ;  ・・0：使用不可  1：使用可能
;                           unsigned BotCnt:8 ;  ・・マウスボタンの数
;                     } ;
;=====================================================================
GetMouseState       proc
                    call    SetVideoInfo
                    MOUSE   0,0
                    xchg    ah,bl
                    ret
GetMouseState       endp
```

カーソルの表示（機能番号：1） ---------------------------

このファンクションは，カーソルの表示を行います．前述のマウスの初期化実行後は，マウスカーソルは非表示となっていますので，このファンクションを実行しなければカーソルは表示されません．また，カーソル表示を実行すると，初回は画面中央に表示されてしまいますので，表示時からカーソル位置を決定しておく必要があれば，当ファンクション実行前に，ファンクション4であらかじめカーソル位置を所定位置へセットしておくとよいでしょう．

当ファンクションのサンプルは，ファンクションの使いやすさを考え，1つにまとめてあります．リスト5.3を参照してください．

MOUSE 0,1

[戻り値]

なし

カーソルの非表示（機能番号：2） -------------------------

このファンクションは，カーソルの非表示（消去）を行います．ある画面領域へ表示，クリアを行うときには，カーソルがその対象となる領域にある可能性があるので，カーソルをいったん非表示にする必要があります．ただし，画面の一部分のカーソルマスク（画面の一部分だけをカーソル表示させたくない）であれば，ファンクションの16を使ったほうがより高速に処理を行うことができます．

また，カーソルは非表示状態でも，マウスを動かせばカーソル位置は動いてしまいますので，カーソル位置を記憶しておく必要があれば，非表示前にカーソル位置をファンクション3で取得し，次のカーソル表示前にファンクション4でカーソル位置をセットしてから，表示するとよいでしょう．

当ファンクションのサンプルは，ファンクションの使いやすさを考え，1つにまとめてあります．リスト5.3を参照してください．

MOUSE 0,2

[戻り値]

なし

カーソル位置とボタン状態の読み取り（機能番号：3） ---------

現在のカーソル位置と，現在のマウスのボタンの状態を読み取ります．通常アプリケーションでは，メニューをマウスで指定させるにせよ，アイコンをマウスで指定させるにせよ，必ずマウスカーソルの位置を把握して置く必要があります．そういった意味ではもっ

リスト5.3　　**カーソルの表示／非表示**

【Cからの呼び出し】

```
{

    CtrlMouseCursor(MC_ON) ;                    // カーソル表示

    CtrlMouseCursor(MC_OFF) ;                   // カーソル非表示

}
```

【MASM6ルーチン】

```
;====================================================================
;       Description : マウスカーソルのON/OFF
;       Sequence    : void  CtrlMouseCursor(int  sw) ;
;       Paramater   : int   sw ; ・・・0：OFF    1：ON
;       戻り値      : void
;====================================================================
CtrlMouseCursor     proc    sw:word
                    mov     ax,sw
                    inc     ax
                    MOUSE
                    ret
CtrlMouseCursor     endp
```

とも頻繁に呼び出されるのが当ファンクションだといってもよいでしょう．

さて，このファンクションを使用するにあたってひとつ注意しなければならないことがあります．このファンクションは，通常キー入力のような「何かキーが押されるまで待って，キーが押されたらファンクションが完了して抜けてくる」タイプのものではありません．したがって，現在位置を取得し，その現在位置に対して画面更新を行う場合などは同じ位置を何度も更新しないように注意する必要があります．リスト5.4に示すCのサンプル関数では，ボタンが押されるか，マウスが移動されるまで，呼び出し元に戻らないようになっています．

MOUSE 0,3

[戻り値]

BL =	現在のボタンの状態		
	ビット0 = 0		左ボタンが押されてない
		= 1	左ボタンが押されている
	ビット1 = 0		右ボタンが押されてない
		= 1	右ボタンが押されている
CX =	現在のカーソルX座標（0からの相対グラフィック座標）		
DX =	現在のカーソルY座標（0からの相対グラフィック座標）		

カーソル位置のセット（機能番号：4）

マウスカーソルの位置を指定座標へセットします．マウスの初期状態は前述したとおりカーソルは画面中央にありますので，必要があればカーソル表示（機能番号1）を行う前に，所定の位置へカーソルをセットしておくとよいでしょう．また，画面更新などを行う

リスト5.4　カーソル位置とボタン状態の読み取り

【Cからの呼び出し】

```
/*************************************************************************
 *************************************************************************
 ***                 マウスカーソル移動すれば戻る                         ***
 *************************************************************************
 *************************************************************************/
GetNezumiPosition(int  StartX,int  StartY,M_CURSOR_INFO *mci,int  atr)
{

    static  int     sx , sy ;

    sx = StartX;
    sy = StartY;

    SetAttribute(1,sy,vinf.Column,atr,REWRITE_ON) ;

    while(1) {
        GetMouseCursorInfo(mci) ;

        // どちらかのボタンが押されたか，移動した場合のみ戻る
        if ( ( !( mci->LeftButton ) && !( mci->RightButton ) ) &&
                        ( sx == mci->CurX && sy == mci->CurY ) )
            continue ;

        ClearAttribute(1,sy,vinf.Column,atr) ;

        return 0 ;
    }

}
```

【MASM6ルーチン】

```
;=========================================================================
;      Description : マウスカーソル位置とボタン状態の読み取り
;      Sequence    : void  GetMouseCursorInfo(struct  MouseCurInfo  *mc) ;
;      Paramater   : struct MouseCursorInfo   *mc ;
;      戻り値      : struct MouseCursorInfo   *mc ;
;=========================================================================
GetMouseCursorInfo  proc    uses    bx cx dx es di,mc:pstr

                    MOUSE   0,3

                    if  LPROG
                        les     di,[mc]
                    else
                    assume  ds:@data
                        MOVSEG  es,ds
                        mov     di,[mc]
                    endif

                    mov     ax,cx                   ; X座標セット
                    add     ax,[divcnt]
                    dec     ax                      ;丸め
                    mov     cl,byte ptr [divcnt]
                    div     cl
                    xor     ah,ah                   ;余りを消す
                    inc     ax
                    stosw

                    mov     ax,dx                   ; Y座標セット
                    add     ax,[divcnt]
                    dec     ax                      ;丸め
                    mov     cx,[divcnt              ;
                    div     cl
                    xor     ah,ah                   ;余りを消す
                    inc     ax
                    stosw

                    mov     ax,bx                   ;左ボタン情報
                    and     ax,1
                    stosb
                    mov     ax,bx                   ;右ボタン情報
                    shr     ax,1
                    and     ax,1
                    stosb

                    ret
GetMouseCursorInfo  endp
```

リスト5.5　　**カーソル位置のセット**

```
【Cからの呼び出し】

{

   SetMouseCursorPos(1,1) ;           // テキストモード時は，1からの相対の
                                      //    テキスト座標で指定

   SetMouseCursorPos(129,1) ;         // グラフィックモード時は，1からの相対の
                                      //    グラフィック座標で指定

}


【MASM6ルーチン】

;===========================================================================
;      Description : マウスカーソル位置設定
;      Sequence    : void  SetMouseCursorPos(int  x,int  y) ;
;      Paramater   : int   x, y ; ・・・・設定するカーソル位置
;      戻り値      : void
;===========================================================================
SetMouseCursorPos   proc    uses    cx dx , x:word , y:word

                    mov     bx,[divcnt]
                    mov     ax,[x]
                    dec     ax
                    mul     bl
                    mov     cx,ax          ; セットするカーソルX座標

                    mov     bx,[divcnt]
                    mov     ax,[y]
                    dec     ax
                    mul     bl
                    mov     dx,ax          ; セットするカーソルY座標

                    MOUSE   0,4

                    ret
SetMouseCursorPos   endp
```

ためにカーソルを非表示にしても，そのあいだにマウスの移動によりカーソル座標は刻々と変化していますので，非表示と再表示時にカーソル位置が変わってしまっては困るような場合，非表示時のカーソル位置を記憶しておき，再表示前に当ファンクションでカーソル位置を戻しておくといった処理にも利用できます（リスト5.5）．

CX =　　　　カーソルのX座標（0からの相対グラフィック座標）
DX =　　　　カーソルのY座標（0からの相対グラフィック座標）
MOUSE 0,4
[戻り値]
なし

ボタンを押した回数と最終位置の読み取り（機能番号：5）-----

指定したボタンの押された回数と，最後にボタンが押されたときの座標，さらに現在のボタンの状態を読み取ります．この機能は，ダブルクリックやドラッグを実現するときに，後述の「ボタンを離した回数と最終位置の読み取り（機能番号：6）」とあわせて，必要となります．MASM6ルーチンをリスト5.6にのせます．

リスト5.8のサンプルプログラムでは「ドラッグ処理」を例にあげて，当ファンクションと「ボタンを離した回数と最終位置の読み取り（機能番号：6）」を併用した例をあげてありますので参考にしてください．

なお，このファンクションは，一度セットされた押された回数については，次回ファンクション発行までクリアされませんので，プログラム中ではクリア用にダミーで一回当ファンクションを実行する必要があります．

BX =　　　　ボタンの指定 0:左　1:右
MOUSE 0,5
[戻り値]
AX =　　　　現在のボタンの状態
　　　ビット0 = 0　左ボタンが押されてない
　　　　　　　= 1　左ボタンが押されている
　　　ビット1 = 0　右ボタンが押されてない
　　　　　　　= 1　右ボタンが押されている
BX =　　　　指定したボタンが押された回数

リスト5.6　　**ボタンを押した回数と最終位置の読み取り**

【MASM6ルーチン】

```
;===========================================================================
;      Description : マウスがクリックされた回数を返す
;      Sequence    : int   GetMouseClickCount(int  b,M_CURSOR_INFO  *mcinf) ;
;      Paramater   : int    b ; ・・・・調べるボタンを指定. 0:左  1:右
;                    M_CURSOR_INFO  *mcinf; 現在のボタンの状態と, ボタン
;                                          が押されたときの座標情報.
;      戻り値      : 指定したボタンの押された回数
;===========================================================================
GetMouseClickCount  proc    uses    bx cx dx , btn:word , mc:pstr
                    mov     bx,[btn]
                    MOUSE   0,5

                    if  @model gt 3
                        les     di, [mc]
                    else
                        mov     di, [mc]
                        MOVSEG  es, ds
                    endif

                    push    ax

                    mov     ax,cx           ; X座標セット
                    add     ax,[divcnt]
                    dec     ax              ;丸め
                    mov     cl,byte ptr [divcnt]
                    div     cl
                    xor     ah,ah           ;余りを消す
                    inc     ax
                    stosw

                    mov     ax,dx           ; Y座標セット
                    add     ax,[divcnt]
                    dec     ax
                    mov     cx,[divcnt]
                    div     cl
                    xor     ah,ah           ;余りを消す
                    inc     ax
                    stosw
                    pop     ax

                    mov     cx,ax           ;現在の左右ボタン情報
                    and     ax,1
                    stosb
                    mov     ax,cx
                    shr     ax,1
                    and     ax,1
                    stosb

                    mov     ax,bx
                    ret
GetMouseClickCount  endp
```

CX =　　最後にボタンが押されたときのX座標（0からの相対グラフィック座標値）

DX =　　最後にボタンが押されたときのY座標（0からの相対グラフィック座標値）

ボタンを離した回数と最終位置の読み取り（機能番号：6）-----

指定したボタンの離された回数と，最後にボタンが離されたときの座標，さらに現在のボタンの状態を読み取ります．この機能は，ダブルクリックやドラッグを実現するときに，前述の「ボタンを押した回数と最終位置の読み取り（機能番号：5）」とあわせて，必要となります．MASM6ルーチンをリスト5.7にのせます．

リスト5.8のサンプルプログラムでは「ドラッグ処理」を例にあげて，当ファンクションと「ボタンを押した回数と最終位置の読み取り（機能番号：5）」を併用した例をあげてありますので参考にしてください．

なお，このファンクションは一度セットされた離された回数については，次回ファンクション発行までクリアされませんので，プログラム中ではクリア用にダミーで一回当ファンクションを実行する必要があります．

BX =　　ボタンの指定 0:左　1:右

MOUSE 0,6

[戻り値]

AX =　　現在のボタンの状態

　　ビット0 = 0　左ボタンが押されてない

　　　　　 = 1　左ボタンが押されている

　　ビット1 = 0　右ボタンが押されてない

　　　　　 = 1　右ボタンが押されている

BX =　　指定したボタンが押された回数

CX =　　最後にボタンが離されたときのX座標（0からの相対グラフィック座標値）

リスト5.7　ボタンを離した回数と最終位置の読み取り

```
【MASM6ルーチン】

;=========================================================================
;      Description : マウスボタンが離された回数を返す
;      Sequence    : int   GetMouseReleaseCount(int  b,M_CURSOR_INFO  *mcinf) ;
;      Paramater   : int   b ; ・・・・調べるボタンを指定. 0:左  1:右
;                    M_CURSOR_INFO  *mcinf; 現在のボタンの状態と, ボタン
;                                           が押されたときの座標情報.
;      戻り値      : 指定したボタンの離された回数
;=========================================================================
GetMouseReleaseCount    proc    uses    bx cx dx , btn:word , mc:pstr
                        mov     bx,[btn]
                        MOUSE   0,6

                        if @model gt 3
                            les     di, [mc]
                        else
                            mov     di, [mc]
                            MOVSEG  es, ds
                        endif

                        push    ax

                        mov     ax,cx           ; X座標セット
                        add     ax,[divcnt]
                        dec     ax
                        mov     cl,byte ptr [divcnt]
                        div     cl
                        xor     ah,ah                   ;余りを消す
                        inc     ax
                        stosw

                        mov     ax,dx           ; Y座標セット
                        add     ax,[divcnt]
                        dec     ax
                        mov     cx,[divcnt]
                        div     cl
                        xor     ah,ah                   ;余りを消す
                        inc     ax
                        stosw
                        pop     ax

                        mov     cx,ax           ;現在の左右ボタン情報
                        and     ax,1
                        stosb
                        mov     ax,cx
                        shr     ax,1
                        and     ax,1
                        stosb

                        mov     ax,bx
                        ret
GetMouseReleaseCount    endp
```

リスト5.8　ドラッグ処理

```
    CtrlMouseCursor(MC_ON) ;                            //マウスカーソル表示

    GetMouseClickCount(MC_LEFTB,&mcinf);                //押された回数クリア
    GetMouseReleaseCount(MC_LEFTB,&mcinf);              //離された回数クリア

    while(1) {
        if ( GetMouseClickCount(MC_LEFTB,&mcinf) ) {    //ボタン押下回数読み取り
            sx = mcinf.CurX ;                           //最後に押された位置を
                                                        //始点とする
            sy = mcinf.CurY ;
            while( !GetMouseReleaseCount(MC_LEFTB,&mcinf) );// 離されるまでループ
            ex = mcinf.CurX ;                           //最後に離された位置を
                                                        //終点とする
            ey = mcinf.CurY ;
            break ;
        }
    }
```

DX =　　　最後にボタンが離されたときのY座標（0からの相対グラフィック座標値）

カーソル移動範囲の設定——X方向（機能番号：7）

X方向のマウスカーソルの移動範囲を設定します．当ファンクション発行時，マウスカーソルが設定範囲を越えた位置にセットされていた場合は，最少（または最大）の境界位置にマウスカーソルは再セットされてしまいます．このため，当ファンクション発行前にカーソルを非表示にし，ファンクション発行後，カーソルを再表示する前にカーソル位置を適当な位置へセットしておくとよいでしょう．

サンプルプログラムでは，当ファンクションと後述の「カーソル移動範囲の設定----Y方向（機能番号：8）」は，ファンクションの使いやすさを考え，1つのルーチンにしてあります．リスト5.9を参照してください．ここでは，最大値の値として，テキストで利用可能な最大値を使用しています．

最小値(CX)，最大値(DX)として指定した値が逆にセットされていた場合は，自動的に内容を入れ替えた状態で移動範囲が設定されます．

CX =　　　カーソルX方向移動範囲最小値

DX =　　　カーソルX方向移動範囲最大値

MOUSE 0,7

[戻り値]

なし

カーソル移動範囲の設定——Y方向（機能番号：8） ----------

Y方向のマウスカーソルの移動範囲を設定します．当ファンクション発行時，マウスカーソルが設定範囲を越えた位置にセットされていた場合は，マウスカーソルは最小（または最大）の境界位置に再セットされてしまいます．このため，当ファンクション発行前にカーソルを非表示にし，ファンクション発行後，カーソルを再表示する前にカーソル位置を適当な位置へセットしておくとよいでしょう．

サンプルプログラムでは，当ファンクションと前述の「カーソル移動範囲の設定----X方向（機能番号：7）」は，ファンクションの使いやすさを考え，1つのルーチンにしてあります．リスト5.9を参照してください．ここでは，最大値の値として，テキストで利用可能な最大値を使用しています．

最小値(CX)，最大値(DX)として指定した値が逆にセットされていた場合は，自動的に内容を入れ替えた状態で移動範囲が設定されます．

CX =　　　　カーソルY方向移動範囲最小値
DX =　　　　カーソルY方向移動範囲最大値
MOUSE 0,8
[戻り値]
なし

グラフィックカーソルの形状設定（機能番号：9） ----------

グラフィックマウスカーソル形状を定義します．グラフィックカーソルは，横16ドット×縦16ラインのビットマップで構成され，ANDマスクパターンと，XORマスクパターンをそれぞれ1つずつもつようになっています（詳細については，本章の「マウスインターフェースの使用上の予備知識」の「マウスカーソル」<P.128>を参照）．さらに，ビットマップで構成されたマウスカーソルには指示点と呼ばれるビットマップ中の1点を指定する必要があります．

BX =　　　　カーソルX指示点（－128～127）
CX =　　　　カーソルY指示点（－128～127）
ES:DX =　　　　ANDパターン，XORパターンのテーブルアドレス

リスト5.9　　**カーソル移動範囲設定**

```
【Cからの呼び出し】

   SetMousePhysBlock(1,2,vinf.Column,vinf.Lines) ; // マウスの移動範囲設定


【MASM6ルーチン】

;==========================================================================
;       Description : マウスの移動範囲設定
;       Sequence    : void  SetMousePhysBlock(int x1,int y1,int x2,int y2) ;
;       Paramater   : int   x1 , y1 ; ・・・移動範囲の最小値
;                     int   x2 , y2 ; ・・・移動範囲の最大値
;       戻り値      : void
;==========================================================================
SetMousePhysBlock   proc    uses bx cx dx , x1:word ,y1:word ,x2:word,y2:word

                    mov     ax, x2
                    dec     ax
                    mov     cx, divcnt
                    mul     cl
                    mov     dx, ax

                    mov     ax, x1
                    dec     ax
                    mov     cx, divcnt
                    mul     cl
                    mov     cx, ax
                    MOUSE   0,7

                    mov     ax, y2
                    dec     ax
                    mov     cx, divcnt
                    mul     cl
                    mov     dx, ax

                    mov     ax, y1
                    dec     ax
                    mov     cx, divcnt
                    mul     cl
                    mov     cx, ax

                    MOUSE   0,8

                    ret
SetMousePhysBlock   endp
```

リスト5.10　グラフィックカーソルの形状定義

【Cからの呼び出し】

```
static  uchar   CursorShip[  = |    // AND パターン
                                    0x3f , 0xff , 0x1f , 0xff , 0x0f , 0xff ,
                                    0x07 , 0xff , 0x03 , 0xff , 0x01 , 0xff ,
                                    0x00 , 0xff , 0x00 , 0x7f , 0x00 , 0x3f ,
                                    0x00 , 0x1f , 0x01 , 0xff , 0x10 , 0xff ,
                                    0x30 , 0xff , 0xf8 , 0x7f , 0xf8 , 0x7f ,
                                    0xfc , 0x7f ,
                                    // XOR パターン
                                    0x00 , 0x00 , 0x40 , 0x00 , 0x60 , 0x00 ,
                                    0x70 , 0x00 , 0x78 , 0x00 , 0x7c , 0x00 ,
                                    0x7e , 0x00 , 0x7f , 0x00 , 0x7f , 0x00 ,
                                    0x7c , 0x00 , 0x6c , 0x00 , 0x46 , 0x00 ,
                                    0x06 , 0x00 , 0x03 , 0x00 , 0x03 , 0x00 ,
                                    0x00 , 0x00 | ;

   SetMouseCursorShape(0,0,CursorShip) ;
```

【MASM6ルーチン】

```
;=====================================================================
;      Description : グラフィックカーソルの定義
;      Sequence    : void  SetMouseCursorShape(int xpoint,int ypoint,char *ship) ;
;      Paramater   : int   xpoint ; ・・・カーソル指示点X
;                    int   ypoint ; ・・・カーソル指示点Y
;                    char  *ship ;  ・・・カーソルビットマップ
;      戻り値      : void
;=====================================================================
SetMouseCursorShape proc    uses    bx cx dx ,  ¥
                                    xpoint:word , ypoint:word , ship:pstr

                    mov     bx,[xpoint]
                    mov     cx,[ypoint]
                    les     dx,[ship]
                    MOUSE   0,9

                    ret
SetMouseCursorShape endp
```

MOUSE 0,9

[戻り値]

なし

このビットマップをいろいろと工夫することによって，指差しカーソルや，時計カーソルなど作成することができるでしょう．リスト5.10のサンプルでは，標準のカーソルパターンを作成する場合のパターンテーブルで登録を行っています．

テキストカーソルの設定（機能番号：10）

テキストカーソルの種類と，その形を定義します．図5.3のテキストカーソルのマスクパターンで説明したとおり，テキストカーソルは2ワード（16ビット×2）のビットパターンで定義します．このビットパターンをもう一度見てみましょう（図5.5）．

右のマスクビットから順に説明していきましょう．一番右は「文字コード」と書いてあります．この情報は現在表示されている文字コードに対して，どういうマスクを行うかを指定するビットになっています．標準のビットパターンでは，ANDマスクは「文字コード」に対して，すべてのビットが1でした．これは，表示文字に対して0でANDを行ってしまうと，その箇所だけ文字が欠けてしまうので，1でANDをとっているのです．次に「前景色」「前景色の強調属性」「背景色」「明滅」これらもすべて1でANDをとっています．「明滅」のビットは日本語モードではサポートされていないとありますが，このビットは「背景色への強調属性」となっており，このビットにより背景色も16色使うことが可能となっています．

このように，基本的にはANDマスクはすべて1，XORマスクの文字コードは0で指定しておけば，XORパターンの色の部分だけを変えれば，自分の好きな色でカーソルを反転表示

テキストカーソルのマスクパターン構成

ビット情報	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
意味	明滅※	背景色			強調	前景色			文字コード							

※ 日本語モードではサポートされていません．

図5.5　テキストカーソルのマスクビット構成

リスト5.11　　テキストカーソルの形状定義

【Cからの呼び出し】

```
	SetMouseCursorMode(MCC_SOFT_CURSOR,0xffff,0x1100) ;
```

【MASM6ルーチン】

```
;=======================================================================
;	Description : テキストカーソルの設定
;	Sequence    : void  SetMouseCursorMode(mode,and_mask,xor_mask) ;
;	Paramater   : int   mode ; ・・・テキストカーソルモード
;	                    0:ソフトウェアカーソル  1:ハードウェアカーソル
;	              unsigned and_mask ; ・・・AND マスク値
;	              unsigned and_mask ; ・・・XOR マスク値
;	戻り値      : void
;=======================================================================
SetMouseCursorMode	proc	uses bx cx dx , cur_mode:word , ¥
				and_mask:word , xor_mask:word

		mov	bx,[cur_mode]
		mov	cx,[and_mask]
		mov	dx,[xor_mask]
		MOUSE	0,10

		ret
SetMouseCursorMode	endp
```

することが可能になるのです．

サンプルをリスト5.11にのせておきます．

BX = 　　カーソルモード
　　0　ソフトウェアカーソル
　　1　ハードウェアカーソル
CX = 　　ANDマスクパターン
DX = 　　XORマスクパターン
MOUSE 0,10
[戻り値]
なし

マウスの移動距離（マウス△）の読み取り（機能番号：11）----

現在設定されているマウスの移動距離（マウスの△）を返します．アプリケーションでこの値を変えるときは，当ファンクションでマウスの△値を保存しておき，アプリケーション終了時に元の値へ戻しておくとよいでしょう．

返される値はX方向，Y方向ともに−32768〜32767までの値になります．

サンプルをリスト5.12にのせておきます．

MOUSE 0,11
[戻り値]
CX= 　　現在設定されているX方向のマウス△
DX= 　　現在設定されているY方向のマウス△

ユーザー割り込みの設定（機能番号：12）------------------

指定したマウスの状態をトリガとして，ユーザー割り込みを発生させます．指定したユ

リスト5.12　　マウスの移動距離の読み取り

【Cからの呼び出し】

```
	int	xdis , ydis ;

	GetMouseMoveDis(&xdis,&ydis) ;
```

【MASM6ルーチン】

```
;=======================================================================
;	Description : マウスの移動距離の読み取り
;	Sequence    : void  GetMouseMoveDis(int  *xdis,int  *ydis) ;
;	Paramater   : int   *xdis ; ・・・X方向のΔの数
;	              int   *ydis ; ・・・Y方向のΔの数
;	戻り値      : void
;=======================================================================
GetMouseMoveDis	proc	uses bx cx dx es di, xdis:pstr , ydis:pstr

		MOUSE	0,11
		.if @model gt 3
		   les	di, xdis
		.else
		   mov	di, xdis
		   MOVSEG	es,ds
		.endif
		mov	ax,cx
		stosw

		.if @model gt 3
		   les	di, ydis
		.else
		   mov	di, ydis
		   MOVSEG	es,ds
		.endif
		mov	ax,dx
		stosw

		ret
GetMouseMoveDis	endp
```

7	6	5	4	3	2	1	0
X	X	X	1	1	1	1	1

CX

- ビット0：カーソル位置が変更された
- ビット1：左ボタンが押された
- ビット2：左ボタンが離れた
- ビット3：右ボタンが押された
- ビット4：右ボタンが離れた

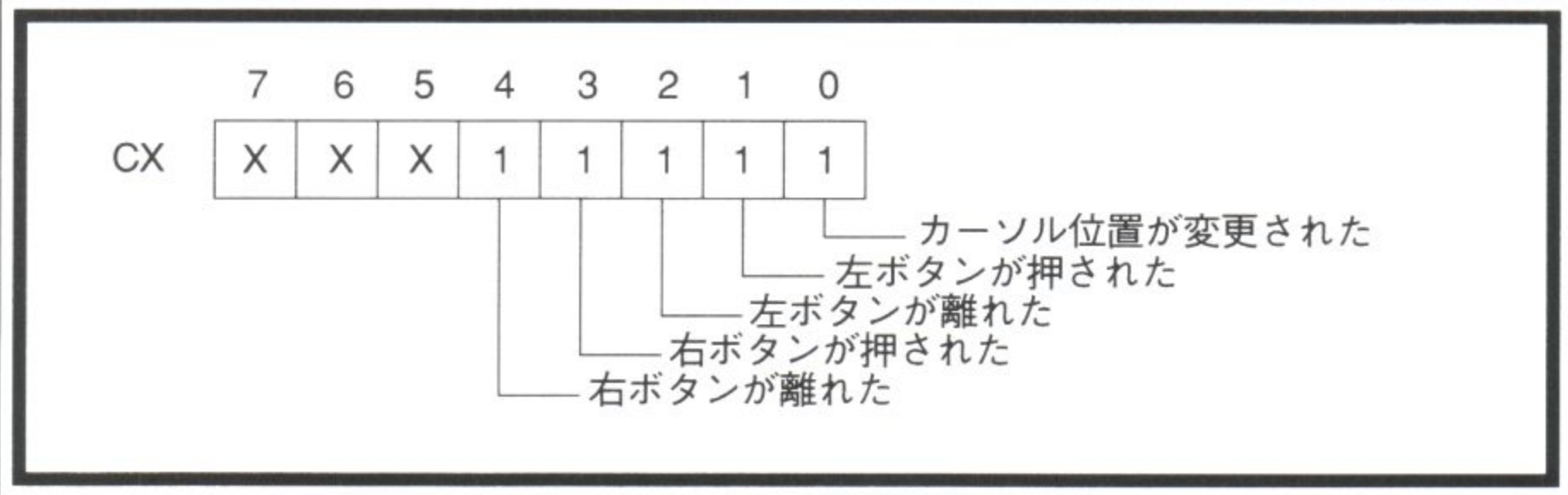



図5.6

ユーザー割り込み用トリガ

ユーザー割り込みが発生したときに，割り込みルーチンに渡される要因と状態

レジスタ	要因と状態
AL	ユーザー割り込みの発生した要因 7 6 5 4 3 2 1 0 X X X 1 1 1 1 1 ビット0：カーソル位置が変更された ビット1：左ボタンが押された ビット2：左ボタンが離れた ビット3：右ボタンが押された ビット4：右ボタンが離れた
BL	ボタンの押下状態 7 6 5 4 3 2 1 0 X X X X X X 1 1 ビット0：左ボタンが押されている ビット1：右ボタンが押されている
CX	カーソルのX座標
DX	カーソルのY座標
SI	X方向のマウスΔ値
DI	Y方向のマウスΔ値

図5.7

割り込みルーチンに渡される要因と状態

ーザー割り込みルーチンは，指定したトリガとなる状態が発生したつど呼び出されます．トリガとして指定できるマウスの状態には図5.6のものがあります．このトリガは複数指定することが可能で，ユーザー割り込みが発生したときに，割り込み要因となるマウス状態がユーザー割り込みルーチンへ渡されます．

ユーザー割り込みルーチンは通常どおり作成してよいものの，DSアドレスはマウスドライバを指してしまいます．このため，Cで作成するユーザー割り込み関数は，ルーチン内からCの標準ライブラリ関数を呼び出すことができません．

図5.7に，ユーザー割り込みルーチンに渡される割り込み要因とマウス状態を示します．

サンプルプログラム（リスト5.13）では，ユーザー割り込み呼び出し用に呼び出し専用割り込みルーチンをもち，自DS内に1024ワードのスタックを確保しています．割り込みルーチンのスタック使用量により，この値を増減するとよいでしょう．

ユーザー割り込みの解放は「マウス機能の初期化（機能番号0）」を行うことにより実行されます．ユーザー割り込みを設定したアプリケーションは，必ずアプリケーション終了前にユーザー割り込みを解放しなければなりません．ただし，アプリケーションの実行中に「マウス機能の初期化」を実行すると，すべての内部変数も初期化されてしまうので注意が必要です．途中でユーザー割り込みを交換したい場合であれば初期化を行うのではなく，「ユーザー割り込みルーチンの差し替え（機能番号20）」を行うとよいでしょう．

サンプルプログラムでは，カーソル移動，右ボタン押下，左ボタン押下の3つをトリガと

して設定し，ユーザー割り込みルーチン内で，割り込み要因を判別し，その要因をメッセージとして表示しています．

CX =　　ユーザー割り込みトリガ

ES:DX =　　ユーザー割り込みルーチンのアドレス

MOUSE 0,12

[戻り値]

なし

ライトペンエミュレーション機能開始（機能番号：13）--------

ライトペンエミュレーションを開始します．当ファンクションを発行すると次回「ライ

リスト5.13　ユーザー割り込み

【Cからの呼び出し】

```
/*************************************************************************/
/*                                                                       */
/*                  マウストリガによる，ユーザー割り込み                 */
/*                                                                       */
/*************************************************************************/
MouseUserInterrupt(void)
{

    CtrlMouseCursor(MC_OFF) ;

    ClearScreen() ;

      PutStringAbs(1,1,"ユーザー割り込みのテストを行います．トリガには移動，左右ボタン
押下があります．",B_ATRB|ATRY|HIGHBRIGHT) ;
                PutStringAbs(1,2,"何かキーを押すと，ユーザー割り込みは解除されます．
",B_ATRB|ATRY|HIGHBRIGHT) ;

    SetMousePhysBlock(1,3,vinf.Column,vinf.Lines) ;

    CtrlMouseCursor(MC_ON) ;

    SetMouseUserInt(MCU_MOVE_CURSOR|MCU_DOWN_LEFTB|MCU_DOWN_RIGHTB,&uf_tst) ;

    while(!kbhit()) ;

    getch() ;

    ReleaseMouseUserInt() ;

    CtrlMouseCursor(MC_OFF) ;

}
/*************************************************************************/
/*                                                                       */
/*                         ユーザー割り込み                              */
/*      注意：この関数では，DS，SS，ESはマウスドライバ内部の             */
/*            セグメントが設定されているので，C標準のライブラリ関数      */
/*            は使用できません．                                         */
/*                                                                       */
/*************************************************************************/
int     uf_tst(unsigned trig,int  left,int  right,int   cx,int  cy, ¥
                                          int  xdis,int  ydis)
{

    static  int     cnt = 0 , yy = 3 ;

    if ( trig & MCU_MOVE_CURSOR )
        PutStringAbs(10,yy++,"カーソルが動きました    ",B_ATRY|B_HIGHBRIGHT|ATRB) ;
    else
    if ( trig & MCU_DOWN_LEFTB )
        PutStringAbs(10,yy++,"左ボタンが押されたました",B_ATRB|HIGHBRIGHT|ATRW) ;
    else
    if ( trig & MCU_DOWN_RIGHTB )
        PutStringAbs(10,yy++,"右ボタンが押されたました",B_ATRM|B_HIGHBRIGHT|ATRG) ;

    if ( yy > ( vinf.Lines - 1 ) ) {
        yy -- ;
        CtrlMouseCursor(MC_OFF) ;
        ScrollUp(1,3,vinf.Column,vinf.Lines-1,1) ;       // マウスクリック範囲
        PutLoopChar(1,vinf.Lines-1,0x20,ATRW,vinf.Column) ; // のスクロール
        ScreenRewrite(3,vinf.Lines-1) ;                      //スクロール範囲
        CtrlMouseCursor(MC_ON) ;                            //マウスカーソル表示
    }

    return 0 ;

}
```

【MASM6ルーチン】

```
;=========================================================================
;       Description : マウストリガ，ユーザー割り込み
;       Sequence    : void  SetMouseUserInt(unsigned flag,int  (*func)()) ;
;       Paramater   : unsigned   flag ; ・・・トリガフラグ
;                     int    (*func)() ; ・・・割り込みプログラム
;       戻り値      : void
;=========================================================================
SetMouseUserInt     proc     uses bx cx dx es , ¥
                                        trig_flag:word , user_func:pstr

                    ; ユーザー関数アドレス待避
                    les      dx,[user_func]
                    mov      word ptr uf_save,dx
                    mov      word ptr uf_save[2],es

                    ; トリガセット
                    mov      cx,[trig_flag]
                    mov      ax,seg  OrigUserFunction
                    mov      es,ax
                    mov      dx,offset OrigUserFunction
                    MOUSE    0,12

                    ret
SetMouseUserInt     endp

;=========================================================================
;       Description : マウストリガ，オリジナルユーザー割り込み
;=========================================================================
OrigUserFunction    proc     private uses    ax bx cx dx ds si es di

                    mov      cs:ax_save,ax

                    mov      ax,@data
                    mov      ds,ax                 ; DSを復元

                    assume   ds:@data

                    mov      cs:ss_save,ss         ; ドライバSSを待避
                    mov      cs:sp_save,sp         ; ドライバSPを待避

                    cli
                    mov      ss,ax                 ; DS = SSに設定
                    mov      sp,offset stack_area
                    sti

                    dec      ax
                    mov      es,ax                 ; ESを設定

                    call     CommonOrigUser

                    push     di
                    push     si
                    push     dx
                    push     cx
                    push     right_b
                    push     left_b
                    push     ax
                    call     [uf_save]
                    add      sp,14

                    cli
                    mov      ax,cs:ss_save
                    mov      ss,ax
                    mov      ax,cs:sp_save
                    mov      sp,ax
                    sti

                    ret
OrigUserFunction    endp
```

リスト5.14 **ライトペンエミュレーション制御**

```
【Cからの呼び出し】

   WritePenEmulation(MWPE_ON) ;          // ﾗｲﾄﾍﾟﾝｴﾐｭﾚｰｼｮﾝ開始
        .
        .
   WritePenEmulation(MWPE_OFF) ;         // ﾗｲﾄﾍﾟﾝｴﾐｭﾚｰｼｮﾝ終了


【MASM6ルーチン】

;===========================================================================
;       Description : ライトペンエミュレーション開始・終了制御
;       Sequence    : void WritePenEmulation(int sw) ;
;       Paramater   : int  sw ; ・・・・0:開始   1:終了
;       戻り値      : void
;===========================================================================
WritePenEmulation   proc    uses    ax , sw:word
                    mov     ax, 13
                    add     ax, sw
                    MOUSE
                    ret
WritePenEmulation   endp
```

トペンエミュレーション機能終了（機能番号：14）」が発行されるまで，マウスはライトペンとして機能するようになります．エミュレーションモードが開始されると，マウスの両方のボタンは，ライトペンを押し下げた状態を表すようになります

サンプルプログラムでは，関数としての使いやすさを考え，1つのルーチンにまとめてあります．リスト5.14を参照してください．

MOUSE 0,13

[戻り値]

なし

ライトペンエミュレーション機能終了（機能番号：14）--------

ライトペンエミュレーションを終了します．

サンプルプログラムでは，関数としての使いやすさを考え，1つのルーチンにまとめてあります．リスト5.14を参照してください．

MOUSE 0,14

[戻り値]

なし

マウスの移動比率の設定（機能番号：15）-----------------

マウスの移動比率（マウスのΔ）を設定します．当ファンクションで移動比率を変更する場合は，「マウスの移動距離（マウスΔ）の読み取り（機能番号：11）」でマウスのΔ値を読み取り保存しておき，アプリケーション終了前に設定前の値に戻しておくとよいでしょう．

このファンクションで指定する値（マウスのΔ）は，約0.5mmを1とした単位になっています．この値は1から32767まで指定可能です．マウスの移動感度が悪い場合などは，この値を大きくして設定すれば，移動感度はよくなります．デフォルトのマウスΔは，X方向が8，Y方向が16に設定されています．

CX =　　X方向のマウスΔ（1～32767）

DX =　　Y方向のマウスΔ（1～32767）

MOUSE 0,15

[戻り値]

なし

カーソル非表示域の設定（機能番号：16）-----------------

カーソルの非表示領域を指定します．このファンクションでは画面の任意の領域に対してカーソル表示を禁止することができます．画面のある領域への表示／クリアを行う場合

リスト5.15　カーソル非表示域の設定

```
【Cからの呼び出し】

   SetMouseClearRange(1,10,1,15) ;      // テキストモードでの指定
                                        // 桁目, 10行目～
                                        // 桁目, 15行目までを非表示に設定


   SetMouseClearRange(1,160,1,240) ;    // グラフィックモードでの指定
                                        // 桁目, 160ライン目～
                                        // 桁目, 240ライン目までを
                                        //非表示に設定


【MASM6ルーチン】

;=========================================================================
;       Description : カーソル消去範囲の設定
;       Sequence    : void  SetMouseClearRange(int x1,int y1,int x2,int y2) ;
;       Paramater   : int   x1 , y1 ; ・・・・消去範囲左上の座標値
;                     int   x2 , y2 ; ・・・・消去範囲右下の座標値
;       戻り値      : void
;=========================================================================
SetMouseClearRange  proc    uses    cx dx si di , ¥
                                    x1:word , y1:word , x2:word , y2:word

                    mov     cx, divcnt
                    mov     ax, y2
                    mul     cl
                    mov     di, ax

                    mov     ax, x2
                    mul     cl
                    mov     si, ax

                    mov     ax, y1
                    mul     cl
                    mov     dx, ax

                    mov     ax, x1
                    mul     cl
                    mov     cx, ax

                    MOUSE   0,16

                    ret
SetMouseClearRange  endp
```

には，表示／クリアを行う対象領域上にカーソルがある可能性があるので，いったんカーソルを非表示にする必要があります．この場合，「カーソルの非表示（機能番号2）」を行うより内部的に処理は高速に行われます．さらに，アプリケーションの実行中，アプリケーション終了までカーソルの非表示領域があるのであれば，このファンクションで指定領域へのカーソル表示を禁止することが可能です．

指定範囲へのカーソル非表示は，「カーソルの表示（機能番号1）」を実行することにより解除されます．

座標の指定はグラフィック座標の0からの相対で指定します．

サンプルをリスト5.15にのせておきます．

CX =　　　　非表示領域の左上のX座標
DX =　　　　非表示領域の左上のY座標
SI =　　　　非表示領域の右下のX座標
DI =　　　　非表示領域の右下のY座標
MOUSE 0,16
[戻り値]
なし

倍速境界値の設定（機能番号：19）――――――――――――

このファンクションでは，マウスの移動量がある一定量に達したときに，画面上のカーソル移動量を2倍にすることが可能です．この境界となる値を倍速境界値と呼び，この値は1秒あたりのマウスの移動量（マウスΔ）で表します．倍速境界値のデフォルト値は64に設定されており，「マウス機能の初期化（機能番号0）」「マウスドライバのソフトウェアリセット（機能番号33）」を呼び出すか，境界値として0を指定すると，この値にリセッ

リスト5.16　倍速境界値の設定

```
【Cからの呼び出し】

   SetMouseOverDrive(0) ;                       // デフォルト倍速境界値を設定


【MASM6ルーチン】

;=========================================================================
;       Description : 倍速境界値の設定
;       Sequence    : void  SetMouseOverDrive(int  OVcnt) ;
;       Paramater   : int   OVcnt ; ・・・倍速境界値
;       戻り値      : void
;=========================================================================
SetMouseOverDrive   proc    uses    dx , OVcnt:word
                    mov     dx, OVcnt
                    MOUSE   0,19
                    ret
SetMouseOverDrive   endp
```

トされます.

サンプルをリスト5.16にのせておきます.

DX =　　　倍速境界値

MOUSE 0,17

[戻り値]

なし

ユーザー割り込みルーチンの差し替え（機能番号：20）--------

「ユーザー割り込みの設定（機能番号12）」で設定したユーザー割り込みルーチンの差し替えを行います. 差し替える内容は, ユーザー割り込み用トリガと, ユーザー割り込みルーチンです. このファンクションを実行すると, すでに設定されていたユーザー割り込み用トリガ, およびユーザー割り込みアドレスが返されます.

リスト5.17のサンプルプログラムでは, 差し替え前のトリガと割り込みアドレスは, 呼び出し元に返すようにはなっていません.

CX =　　　新しいユーザー割り込み用トリガ

ES:DX =　　　新しいユーザー割り込みルーチンアドレス

MOUSE 0,18

[戻り値]

CX =　　　オリジナルのユーザー割り込み用トリガ

ES:DX =　　　オリジナルのユーザー割り込みルーチンアドレス

ドライバ状態保管用バッファサイズの取得（機能番号：21）----

マウスを使用している別のプログラムへ一時的に割り込み制御を移す場合は, 現在使用しているマウスドライバ状態保管バッファを待避する必要があります. このとき, ドライバ状態保管用バッファとして必要なサイズを当ファンクションで得ることができます. 実際のドライバ状態を保管するには, 後述の「ドライバ状態の保管（機能番号22）」を使用します.

なお, リスト5.18のサンプルプログラムでは, ドライバ状態保管ルーチンとして1つにまとめています. このルーチンは内部で当該ファンクションを発行し, 必要なサイズ分のメモリを取得して保管を行っています. ここで取得したメモリは, ドライバ状態の復元を行

リスト5.17　ユーザー割り込みルーチンの差し替え

【Cからの呼び出し】

```
{
        ResetMouseUserInt(MCU_MOVE_CURSOR|MCU_DOWN_LEFTB,&uf_newfunc) ;
}
int     uf_newfunc(unsigned trig,int  left,int  right,int   cx,
                                        int  cy,int  xdis,int  ydis)
{
             .
             .
             .
}
```

【MASM6ルーチン】

```
;==============================================================================
;      Description : ユーザー割り込みのマスクと割り込みルーチンの交換
;      Sequence    : void  ResetMouseUserInt(unsigned flag,int  (*func)()) ;
;      Paramater   : unsigned  flag ; ・・・トリガフラグ
;                    int   (*func)() ;・・・ユーザー割り込みルーチン
;      戻り値      : void
;==============================================================================
ResetMouseUserInt   proc    uses    es dx cx ,trig_flag:word , user_func:pstr

                    ;  ユーザー関数アドレス待避
                    les     dx,[user_func]
                    mov     word ptr uf_save,dx
                    mov     word ptr uf_save[2],es

                    ;  トリガセット
                    mov     cx,[trig_flag]
                    mov     ax,seg  OrigUserFunction
                    mov     es,ax
                    mov     dx,offset OrigUserFunction
                    MOUSE   0,20

                    ret
ResetMouseUserInt   endp
```

わないと解放されません.

MOUSE 0,21

[戻り値]

BX =　　　　保管に必要なバッファサイズ

ドライバ状態の保管（機能番号：22）―――――――――――――――――――――

マウスドライバの状態を保管します.

リスト5.18のサンプルプログラムでは，SaveMouseInfo()というルーチン内で，保管用に「保管に必要なバッファサイズの取得」と「ドライバ状態の保管」を行っています.

ES:DX =　　　保管用バッファアドレス

MOUSE 0,22

[戻り値]

なし

ドライバ状態の復元（機能番号：23）―――――――――――――――――――――

前述の「ドライバ状態の保管（機能番号：22）」で保管した内容を復元します.

リスト5.18のサンプルプログラムでは，RestoreMouseInfo()というルーチン内で，復元用にSaveMouseInfo()で確保したバッファの解放を行っています.

ES:DX =　　　保管用バッファアドレス

MOUSE 0,23

[戻り値]

なし

リスト5.18に，マウスドライバ状態の保管と，復元を行うサンプルプログラムをのせます.

リスト5.18　ドライバ状態の保管／復元

```
【Cからの呼び出し】

|

        if ( SaveMouseInfo() ) |            // ドライバ状態の保管
            printf("保管用のバッファが確保できませんでした. ¥7¥n") ;
            exit(-1) ;
        |

        system("UseMouse.exe") ;            // 他のマウス使用プログラムを起動

        RestoreMouseInfo() ;                // ドライバ状態の復元
|


【MASM6ルーチン】

;=====================================================================
;          Description : マウスドライバの状態保存
;          Sequence    : int SaveMouseInfo(void) ;
;          Paramater   : void
;          戻り値       :  0: 正常終了
;                         -1: Not enough memory
;=====================================================================
SaveMouseInfo          proc   uses   bx dx es

                MOUSE 0,21                      ; 状態保存用に必要なバッファサイズ

                add    bx,15
                shr    bx,1
                shr    bx,1
                shr    bx,1
                shr    bx,1

                MSDOS 48h                       ; 記憶域の割り当て

                .if !carry?                     ; 正常終了?
                       mov    [InfoSeg],ax      ; セグメント保存
                       mov    es,ax             ; 保存バッファアドレス設定
                       xor    dx,dx             ; オフセット設定
                       MOUSE 0,22
                       xor    ax,ax             ; 戻り値=0
                .else
                       mov    ax,-1             ; 戻り値=-1
                .endif

                ret
SaveMouseInfo          endp

;=====================================================================
;          Description : マウスドライバの状態復帰
;          Sequence    : void      RestoreMouseInfo(void) ;
;          Paramater   : void
;          戻り値       :  0: 正常終了
;                    -1: Anallocated memory
;=====================================================================
RestoreMouseInfo proc  uses   dx es

                mov    ax,-1

                .if [InfoSeg  != -1
                       mov    es,[InfoSeg]      ; Mouse Info. buffer
                       xor    dx,dx             ; address set
                       MOUSE 0,23

                       MSDOS 49h                ; Release Allocate Block

                       xor    ax,ax             ; InfoSeg reset to -1
                       dec    ax
                       mov    [InfoSeg],ax
                       inc    ax
                .endif

                ret
RestoreMouseInfo endp
```

代替ユーザー割り込みルーチンの設定（機能番号：24） --------

指定したマウスの状態と，キーボードのシフト状態をトリガとして，ユーザー割り込みを発生させます．指定したユーザー割り込みルーチンは，指定したトリガとなる状態が発生したつど呼び出されます．トリガとして指定できるマウスの状態には以下のものがあります．

このトリガは複数指定することが可能で，すべてのボタン操作トリガに対して[Ctrl]，[Alt]，[Shift]キーをそれぞれ組み合せて指定できます（図5.8）．また，ユーザー割り込みは異なるトリガのパターンに対して，同時に最大3つまでのユーザー割り込みルーチンを設定することが可能で，それらは指定したトリガの検知により，非同期に割り込みを発生させます．指定されたトリガのユーザー割り込みが発生すると，割り込み要因となるマウス状態がユーザー割り込みルーチンへ渡されます．

基本的な使用方法は「ユーザー割り込みの設定（機能番号12）」と同じですが，その機能と異なる点は，このファンクションは他のアプリケーションと資源を共有することを前提に作ったと思われる点でしょう．このため，このファンクションを使う場合は約束ごととして，ユーザー割り込みルーチンを設定する前に，後述の「代替ユーザー割り込みルーチンのアドレス取得（機能番号25）」で，指定した割り込みトリガに対して設定されているユーザー割り込みルーチンのアドレスを取得保管しておかなければなりません．そして該当する割り込みトリガに対して設定されているユーザー割り込みが不要になったならば，取得保管しておいたユーザー割り込みルーチンに，当ファンクションを使用して再設定しなければなりません．

ユーザー割り込みルーチンは通常どおり作成してよいものの，DSアドレスはマウスドライバを指してしまいます．このため，Cで作成するユーザー割り込み関数は，ルーチン内からCの標準ライブラリ関数を呼び出すことができません．

図5.9に，代替ユーザー割り込みルーチンに渡される割り込み要因とマウス状態を示します．通常のユーザー割り込みと同じく，返される要因にはトリガとなった制御キーのビットは通知されません．必要があれば，割り込みルーチン内でシフトステータスを検知しなければなりません．

代替ユーザー割り込みの解放は「ユーザー割り込みの設定（機能番号12）」で設定したユーザー割り込みの解放のしかたとは異なり，ファンクションの発行によって解放されるものではありません．代替ユーザー割り込みは，ユーザー割り込み設定前に「ドライバ状態の保管（機能番号22）」を行い，代替ユーザー割り込みが不要になったときに「ドライ

	7	6	5	4	3	2	1	0
CX	1	1	1	1	1	1	1	1

- ビット0：カーソル位置が変更された
- ビット1：左ボタンが押された
- ビット2：左ボタンが離れた
- ビット3：右ボタンが押された
- ビット4：右ボタンが離れた
- ビット5：ボタンの操作と同時に[Shift]キーが押された
- ビット6：ボタンの操作と同時に[Ctrl]キーが押された
- ビット7：ボタンの操作と同時に[Alt]キーが押された

図5.8

代替ユーザー割り込み用トリガ

図5.9

代替割り込みルーチンに渡される要因と状態

代替ユーザー割り込みが発生したときに，割り込みルーチンに渡される要因と状態

レジスタ	要因と状態
AL	ユーザー割り込みの発生した要因 7 6 5 4 3 2 1 0 X X X 1 1 1 1 1 0: カーソル位置が変更された 1: 左ボタンが押された 2: 左ボタンが離れた 3: 右ボタンが押された 4: 右ボタンが離れた
BL	ボタンの押下状態 7 6 5 4 3 2 1 0 X X X X X X 1 1 0: 左ボタンが押されている 1: 右ボタンが押されている
CX	カーソルのX座標
DX	カーソルのY座標
SI	X方向のマウスΔ値
DI	Y方向のマウスΔ値

バ状態の復元（機能番号23）」を実行することでのみ解放を行うことが可能です．また，代替ユーザー割り込みは，「ユーザー割り込みルーチンの差し替え（機能番号20）」での割り込みルーチンの差し替えも，行うことができません．

リスト5.19のサンプルプログラムでは，カーソル移動，右ボタン押下＋（[Alt],[Ctrl],[Shift]キー）をトリガとして設定し，制御キーそれぞれに，ユーザー割り込み1，ユーザー割り込み2，ユーザー割り込み3を設定しています．指定したトリガを検知すると，3つの割り込みルーチンは非同期に割り込みを発生させます．ユーザー割り込みルーチン内では，割り込み要因を16進数で表示させ，何番のユーザー割り込みが発生したかメッセージを表示しています．

CX = ユーザー割り込み用トリガ
ES:DX = ユーザー割り込みルーチンのアドレス
MOUSE 0,24
[戻り値]
なし

代替ユーザー割り込みルーチンのアドレスの取得（機能番号：25） –

「代替ユーザー割り込みルーチンの設定（機能番号：24）」で設定された，ユーザー割り込みルーチンのアドレスを取得します．この機能は，代替ユーザー割り込み設定でユーザー割り込みルーチンを設定する前に使用します．設定するトリガに対して，すでに設定されているユーザー割り込みルーチンがあれば，そのユーザー割り込みルーチンのアドレスを取得して保管する必要があります．詳細については，前述の「代替ユーザー割り込みルーチンの設定（機能番号：24）」を参照してください．

CX = ユーザー割り込み用トリガ

MOUSE 0,25

[戻り値]

なし

リスト5.19

【Cからの呼び出し】

```
/****************************************************************************/
/*                                                                          */
/*                  マウストリガによる，代替ユーザー割り込み                */
/*                                                                          */
/****************************************************************************/
MouseUserInterrupt2(void)
{

    CtrlMouseCursor(MC_OFF) ;                          //マウスカーソルOFF

    ClearScreen() ;                                    //画面消去

    PutStringAbs(1,1,"代替ユーザー割り込みのテストを行います．"
              "トリガには移動，左ボタン+（ALT,CTRL,SHIFTキー）があります．",
                                         B_ATRB|ATRY|HIGHBRIGHT) ;

    PutStringAbs(1,2,"何かキーを押すと，ユーザー割り込みは解除されます．",
                                         B_ATRB|ATRY|HIGHBRIGHT) ;

    SetMousePhysBlock(1,3,vinf.Column,vinf.Lines) ;    // マウスカーソル移動範囲の
                                                       //設定

    CtrlMouseCursor(MC_ON) ;                           //マウスカーソルON

    SaveMouseInfo() ;                                  //マウスドライバ状態の保存

    // ユーザー割り込み1設定
    SetExtMouseUserInt(MCU_MOVE_CURSOR|MCU_DOWN_LEFTB|MCUE_SHIFT_KEY,&uf_tst1,0) ;

    // ユーザー割り込み2設定
    SetExtMouseUserInt(MCU_MOVE_CURSOR|MCU_DOWN_LEFTB|MCUE_CTRL_KEY,&uf_tst2,1) ;

    // ユーザー割り込み3設定
    SetExtMouseUserInt(MCU_MOVE_CURSOR|MCU_DOWN_LEFTB|MCUE_ALT_KEY,&uf_tst3,2) ;

    while(!kbhit()) ;                                  //キーが押されるまで繰り返す

    getch() ;

    ReleaseExtMouseUserInt(0) ;                        //ユーザー割り込み1を戻す
    ReleaseExtMouseUserInt(1) ;                        //ユーザー割り込み2を戻す
    ReleaseExtMouseUserInt(2) ;                        //ユーザー割り込み3を戻す

    RestoreMouseInfo() ;                               //マウスドライバ状態を復元

    CtrlMouseCursor(MC_OFF) ;                          //マウスカーソルON

}
/****************************************************************************/
/*                                                                          */
/*                           代替ユーザー割り込み1                          */
/*                                                                          */
/****************************************************************************/
int     uf_tst1(unsigned trig,int  left,int  right,int   ccx,int  ccy, ¥
                                                int  xdis, int  ydis)
{

    static  int     cnt = 0 , yy = 3 ;

    PutStringAbs(20,yy++,PutHex("ユーザー割り込み1を検知しました．"
                             "割り込みトリガ[%]",trig),ATRW) ;

    if ( yy > ( vinf.Lines - 1 ) ) {
        yy -- ;
        CtrlMouseCursor(MC_OFF) ;                          //マウスカーソル非表示
        ScrollUp(1,3,vinf.Column,vinf.Lines-1,1) ;         // マウスクリック範囲
        PutLoopChar(1,vinf.Lines-1,0x20,ATRW,vinf.Column) ; // のスクロール
        ScreenRewrite(3,vinf.Lines-1) ;                    //スクロール範囲
        CtrlMouseCursor(MC_ON) ;                           //マウスカーソル表示
    }

    return 0 ;

}
/****************************************************************************/
/*                                                                          */
/*                           代替ユーザー割り込み2                          */
/*                                                                          */
/****************************************************************************/
int     uf_tst2(unsigned trig,int  left,int  right,int   ccx,int  ccy, ¥
                                                int  xdis, int  ydis)
{

    static  int     cnt = 0 , yy = 3 ;

    PutStringAbs(20,yy++,PutHex("ユーザー割り込み2を検知しました．"
                             "割り込みトリガ[%]",trig),ATRC) ;

    if ( yy > ( vinf.Lines - 1 ) ) {
        yy -- ;
        CtrlMouseCursor(MC_OFF) ;                          //マウスカーソル非表示
        ScrollUp(1,3,vinf.Column,vinf.Lines-1,1) ;         // マウスクリック範囲
        PutLoopChar(1,vinf.Lines-1,0x20,ATRW,vinf.Column) ; // のスクロール
        ScreenRewrite(3,vinf.Lines-1) ;                    //スクロール範囲
        CtrlMouseCursor(MC_ON) ;                           //マウスカーソル表示
    }

    return 0 ;

}
/****************************************************************************/
/*                                                                          */
/*                           代替ユーザー割り込み3                          */
/*                                                                          */
/****************************************************************************/
int     uf_tst3(unsigned trig,int  left,int  right,int   ccx,int  ccy, ¥
                                                int  xdis, int  ydis)
{

    static  int     cnt = 0 , yy = 3 ;

    PutStringAbs(20,yy++,PutHex("ユーザー割り込み3を検知しました．"
                             "割り込みトリガ[%]",trig),ATRY) ;

    if ( yy > ( vinf.Lines - 1 ) ) {
        yy -- ;
        CtrlMouseCursor(MC_OFF) ;                          //マウスカーソル非表示
        ScrollUp(1,3,vinf.Column,vinf.Lines-1,1) ;         // マウスクリック範囲
        PutLoopChar(1,vinf.Lines-1,0x20,ATRW,vinf.Column) ; // のスクロール
        ScreenRewrite(3,vinf.Lines-1) ;                    //スクロール範囲
        CtrlMouseCursor(MC_ON) ;                           //マウスカーソル表示
    }

    return 0 ;

}
/****************************************************************************/
/*                                                                          */
/*                              4桁Hex表示                                  */
/*                                                                          */
/****************************************************************************/
static  uchar   far *PutHex(char  *ptr,unsigned  hex)
{

    int     i = 0 ;
    static  uchar   buff[128  ;

    while(1) {
        buff[i++  = *(ptr++) ;
        if ( buff[i-1  == '%' ) {
            i -- ;
            buff[i++  = (char)( ( hex >> 12 ) +
                             ( ( hex >> 12 ) > 9 ? 0x40 : 0x30 ) ) ;
            buff[i++  = (char)( ( ( hex >> 8 ) & 0x00ff ) +
                             ( ( ( hex >> 8 ) & 0x00ff ) > 9 ? 0x40 : 0x30 ) ) ;
            buff[i++  = (char)( ( ( hex >> 4 ) & 0x000f ) +
                             ( ( ( hex >> 4 ) & 0x000f ) > 9 ? 0x40 : 0x30 ) ) ;
            buff[i++  = (char)( ( hex & 0x000f ) +
                             ( ( hex & 0x000f ) > 9 ? 0x40 : 0x30 ) ) ;
            buff[i++  = 'H' ;
            continue ;
        }
        if ( buff[i-1  == 0x00 ) break ;
    }

    return buff ;

}
```

【MASM6ルーチン】

```
;==========================================================================
;       Description : 代替割り込みルーチンの設定
;       Sequence    : void  SetExtMouseUserInt(unsigned flag,int  (*func)()) ;
;       Paramater   : unsigned  flag ; ・・・トリガフラグ
;                     int    (*func)() ;・・・ユーザー割り込みルーチン
;                     int    user_no ;  ・・・ユーザー番号（0～2）
;       戻り値      : void
;==========================================================================
SetExtMouseUserInt  proc    uses    bx cx dx es di , trig_flag:word , ¥
                                            user_func:pstr , user_no:word

                    mov     cx, trig_flag
                    MOUSE   0,25                ; 代替割り込みアドレスの保管

                    mov     ax,cx               ; cx待避

                    mov     cx, user_no
                    shl     cx, 1
                    shl     cx, 1
                    shl     cx, 1               ; user_no * 8
```

```
                    ; 現在設定されているトリガに対する割り込みを保管
                    mov     di, offset orig_uf_save
                    add     di, cx
                    mov     (orig_uf ptr [di]).trigger, ax
                    mov     (orig_uf ptr [di]).orig_uf_off, dx
                    mov     (orig_uf ptr [di]).orig_uf_seg, bx

                    mov     cx, user_no
                    shl     cx,1
                    shl     cx,1                ; user_no * 4

                    ; ユーザー割り込みアドレスの待避
                    les     dx, user_func
                    mov     di, offset uf_save1
                    add     di, cx
                    mov     [di], dx
                    mov     [di+2], es

                    ; ユーザー割り込みの設定
                    mov     di, offset uf
                    add     di, cx
                    mov     dx, [di]
                    mov     ax, [di+2]
                    mov     es, ax

                    ; 割り込みトリガを設定
                    mov     cx, trig_flag
                    MOUSE   0,24

                    ret
SetExtMouseUserInt  endp

;=====================================================================
;       Description : マウストリガ, オリジナルユーザー割り込み1
;=====================================================================
OrigUserFunction1   proc    private uses ax bx cx dx ds si es di

                    mov     cs:ax_save, ax

                    mov     ax, @data
                    mov     ds, ax              ; DSを復元
                    assume  ds:@data

                    mov     cs:ss_save, ss      ;ドライバSSを待避
                    mov     cs:sp_save, sp      ;ドライバSPを待避

                    cli
                    mov     ss, ax              ; DS = SSに設定
                    mov     sp, offset stack_area
                    sti

                    dec     ax
                    mov     es, ax              ; ESを設定

                    call    CommonOrigUser

                    push    di
                    push    si
                    push    dx
                    push    cx
                    push    right_b
                    push    left_b
                    push    ax
                    call    uf_save1
                    add     sp, 14

                    cli
                    mov     ax, cs:ss_save
                    mov     ss, ax
                    mov     ax, cs:sp_save
                    mov     sp, ax
                    sti

                    ret
OrigUserFunction1   endp

;=====================================================================
;       Description : マウストリガ, オリジナルユーザー割り込み2
;=====================================================================
OrigUserFunction2   proc    private uses ax bx cx dx ds si es di

                    mov     cs:ax_save, ax

                    mov     ax, @data
                    mov     ds, ax              ; DSを復元
                    assume  ds:@data

                    mov     cs:ss_save, ss      ;ドライバSSを待避
                    mov     cs:sp_save, sp      ;ドライバSPを待避

                    cli
                    mov     ss, ax              ; DS = SSに設定
                    mov     sp, offset stack_area
                    sti

                    dec     ax
                    mov     es, ax              ; ESを設定

                    call    CommonOrigUser

                    push    di
                    push    si
                    push    dx
                    push    cx
                    push    right_b
                    push    left_b
                    push    ax
                    call    uf_save2
                    add     sp,14

                    cli
                    mov     ax, cs:ss_save
                    mov     ss, ax
                    mov     ax, cs:sp_save
                    mov     sp, ax
                    sti

                    ret
OrigUserFunction2   endp

;=====================================================================
;       Description : マウストリガ, オリジナルユーザー割り込み3
;=====================================================================
OrigUserFunction3   proc    private uses ax bx cx dx ds si es di

                    mov     cs:ax_save, ax

                    mov     ax, @data
                    mov     ds, ax              ; DSを復元
                    assume  ds:@data

                    mov     cs:ss_save, ss      ;ドライバSSを待避
                    mov     cs:sp_save, sp      ;ドライバSPを待避

                    cli
                    mov     ss, ax              ; DS = SSに設定
                    mov     sp, offset stack_area
                    sti

                    dec     ax
                    mov     es, ax              ; ESを設定

                    call    CommonOrigUser

                    push    di
                    push    si
                    push    dx
                    push    cx
                    push    right_b
                    push    left_b
                    push    ax
                    call    uf_save3
                    add     sp, 14

                    cli
                    mov     ax, cs:ss_save
                    mov     ss, ax
                    mov     ax, cs:sp_save
                    mov     sp, ax
                    sti

                    ret
OrigUserFunction3   endp

;=====================================================================
;       Description : オリジナルユーザー割り込み共通部分
;=====================================================================
CommonOrigUser      proc    near

                    mov     ax, cs:ax_save

                    push    ax

                    push    cx
                    mov     ax, dx              ; Y座標セット
                    add     ax, divcnt
                    dec     ax
                    mov     cx, divcnt
                    div     cl
                    xor     ah, ah              ;余りを消す
                    inc     ax
                    mov     dx, ax
                    pop     cx

                    mov     ax, cx              ; X座標セット
                    add     ax, divcnt
                    dec     ax
                    mov     cx,[divcnt]
                    div     cl
                    xor     ah, ah              ;余りを消す
                    inc     ax
                    mov     cx, ax

                    xor     ax, ax
                    rcr     bx, 1
                    adc     al, 0
                    mov     left_b, ax

                    xor     ax, ax
                    rcr     bx, 1
                    adc     al, 0
                    mov     right_b, ax

                    pop     ax

                    ret
CommonOrigUser      endp
```

```
;=======================================================================
;      Description : 代替割り込みルーチンの復元
;      Sequence    : void  ReleaseExtMouseUserInt(int  UserNo) ;
;      Paramater   : int   user_no ;  ・・・ユーザー番号 (0~2)
;      戻り値      : void
;=======================================================================
ReleaseExtMouseUserInt      proc    uses bx cx dx, user_no:word

                 mov     cx, user_no
                 shl     cx, 1
                 shl     cx, 1
                 shl     cx, 1                ; user_no * 8
                 mov     di, offset orig_uf_save
                 add     di, cx

                 mov     cx, (orig_uf ptr [di]).trigger
                 mov     dx, (orig_uf ptr [di]).orig_uf_off
                 mov     es, (orig_uf ptr [di]).orig_uf_seg
                 MOUSE   0,24

                 ret
ReleaseExtMouseUserInt      endp
```

マウス感度の設定（機能番号：26）――――――――――――

マウスの移動比率，倍速境界値を調整するための係数を設定します．当ファンクションで実際の移動比率や倍速境界値を設定するわけではなく，ここで設定された係数をもとに演算を行い，演算結果を実際の移動比率や倍速境界値として使用します．この係数は初期値が50に設定されており，50以下を設定すると演算後の値は小さくなり，50以上であれば演算後の値は大きくなります．係数として指定できるのは1～100までの範囲です．

BX =　　X方向の移動比率演算用係数
CX =　　Y方向の移動比率演算用係数
DX =　　倍速境界値演算用係数
MOUSE 0,26
[戻り値]
なし

リスト5.20　　**マウス感度の設定**

```
【Cからの呼び出し】

    SetMouseCoefficient(60,60,100) ;              // X 移動比率演算係数=60
                                                  // Y移動比率演算係数=60
                                                  //倍速境界値演算係数=100

【MASM6ルーチン】

;=======================================================================
;      Description : マウス感度の設定
;      Sequence    : void  SetMouseCoefficient(int xcoef,int ycoef,int ovcoef) ;
;      Paramater   : int   xcoef ; ・・・X方向移動比率の係数(1-100)
;                    int   ycoef ; ・・・Y方向移動比率の係数(1-100)
;                    int   ovcoef ;・・・倍速境界値の係数(1-100)
;      戻り値      : void
;=======================================================================
SetMouseCoefficient proc     uses    bx cx dx ,       ¥
                                     xcoef:word , ycoef:word , ovcoef:word

                 mov     bx,[xcoef]
                 mov     cx,[ycoef]
                 mov     dx,[ovcoef]
                 MOUSE   0,26

                 ret
SetMouseCoefficient endp
```

マウス感度の取得（機能番号：27）――――――――――――

設定されているマウスの移動比率，倍速境界値を調整するための係数を取得します．

MOUSE 0,27
[戻り値]
BX =　　X方向の移動比率演算用係数
CX =　　Y方向の移動比率演算用係数
DX =　　倍速境界値演算用係数

リスト5.21　　**マウス感度の取得**

```
【Cからの呼び出し】

    int      xccoef ;                              // X 移動比率演算係数
    int      yccoef ;                              // Y 移動比率演算係数
    int      ovcoef ;                              //倍速境界値演算係数

    GetMouseCoefficient(&xccoef,&yccoef,&ovcoef) ;

【MASM6ルーチン】

;=======================================================================
;      Description : マウス感度の取得
;      Sequence    :
;          void    GetMouseCoefficient(int *xcoef,int *ycoef,int *ovcoef) ;
;      Paramater   : int   *xcoef ; ・・・X方向移動比率の係数(1-100)
;                    int   *ycoef ; ・・・Y方向移動比率の係数(1-100)
;                    int   *ovcoef ;・・・倍速境界値の係数(1-100)
;      戻り値      : void
;=======================================================================
GetMouseCoefficient proc     uses    bx cx dx es di ,        ¥
                                     xcoef:pstr , ycoef:pstr , ovcoef:pstr

                 MOUSE   0,26

                 if @model gt 3
                    les     di, xcoef
                 else
                    mov     di, xcoef
                    MOVSEG  es, ds
```

```
        endif
        mov     ax,bx
        stosw

        if @model gt 3
            les     di, ycoef
        else
            mov     di, ycoef
        endif
        mov     ax,cx
        stosw
```

```
        if @model gt 3
            les     di, ovcoef
        else
            mov     di, ovcoef
        endif
        mov     ax,dx
        stosw

        ret
GetMouseCoefficient endp
```

カーソル表示ページの設定（機能番号29）

マウスカーソルを表示する画面ページを設定します．DOS/V日本語モードでは，現在，画面ページはサポートされていませんので，0を設定します．また，グラフィックモード，DOS/V英語モードはグラフィックカードに依存しますので，グラフィックカードのマニュアル，および本書の「第2章　画面制御編」を参照してください．

BX =　　画面ページ番号

MOUSE 0,29

[戻り値]

なし

リスト5.22　　**カーソル表示ページの設定**

```
【Cからの呼び出し】

    SetMouseCrtPage(0) ;                    //画面ページを0に設定

【MASM6ルーチン】

;=========================================================================
;       Description : 画面ページ番号の設定
;       Sequence    : void  SetMouseCrtPage(int  crt_page) ;
;       Paramater   : int   crt_page ; ・・・カーソルを表示する画面ページ
;       戻り値      : void
;=========================================================================
SetMouseCrtPage     proc    uses bx, crt_page:word

                    mov     bx, crt_page
                    MOUSE   0,29

                    ret
SetMouseCrtPage     endp
```

カーソル表示ページの取得（機能番号30）

マウスカーソルが表示されるCRTページを取得します．

MOUSE 0,30

[戻り値]

BX =　　画面ページ番号

リスト5.23　　**カーソル表示ページの取得**

```
【Cからの呼び出し】

    int     crt_page = GetMouseCrtPage() ;

【MASM6ルーチン】

;=========================================================================
;       Description : 画面ページ番号の取得
;       Sequence    : int   GetMouseCrtPage(void) ;
;       Paramater   : void
;       戻り値      : int   crt_page ; ・・・カーソルが表示される画面ページ
;=========================================================================
GetMouseCrtPage     proc    uses bx
                    MOUSE   0,30
                    mov     ax,bx
                    ret
GetMouseCrtPage     endp
```

マウスドライバの使用禁止設定（機能番号31）

マウスドライバ割り込みを使用禁止にします．マウスドライバは割り込み33hのほかに，割り込み10H，割り込み71H（i286,i386,i486の場合は74H）をフックしています．

当ファンクションは，この33hを除くすべてのベクタの状態を，マウスドライバが組み込みされる前の状態へ戻します．マウスドライバが使用しているベクタのうち，1つでも解放できないベクタがあると，当ファンクションは正常終了しません．この場合，戻り値として−1を返します．

BX =　　　　元の割り込み33Hベクタに設定されていたオフセット

CX =　　　　元の割り込み33Hベクタに設定されていたセグメント

MOUSE 0,31

[戻り値]

なし

リスト5.24　マウスドライバの使用禁止設定

```
【Cからの呼び出し】

    if ( 0 > ReleaseMouseDriver() ) {
        printf("マウスドライバの使用するベクタを解放できませんでした．¥n") ;
        exit(-1) ;
    }


【MASM6ルーチン】

;==========================================================================
;       Description : マウスドライバのメモリからの消去
;       Sequence    : int   ReleaseMouseDriver(void) ;
;       Paramater   : void
;       戻り値      : 0: 正常終了
;                     -1: 異常終了
;==========================================================================
ReleaseMouseDriver  proc    uses bx cx

                    mov     bx,word ptr [33h*4]
                    mov     cx,word ptr [33h*4][2]
                    MOUSE   0,31

                    ret
ReleaseMouseDriver  endp
```

マウスドライバの使用禁止解除（機能番号32）----------------

「マウスドライバの使用禁止設定（機能番号31）」で使用禁止にしたマウスドライバを再使用可能にします．このファンクションでは，さきのファンクションで解放されたベクタを再設定します．

MOUSE 0,32

[戻り値]

なし

リスト5.25　マウスドライバの使用禁止解除

```
【Cからの呼び出し】

    ResetMouseDriver() ;


【MASM6ルーチン】

;==========================================================================
;       Description : マウスドライバのメモリへの再設定
;       Sequence    : void  ResetMouseDriver(void) ;
;       Paramater   : void
;       戻り値      : void
;==========================================================================
ResetMouseDriver    proc
                    MOUSE   0,32
                    ret
ResetMouseDriver    endp
```

マウスドライバのソフトウェアリセット（機能番号33）--------

マウスドライバが使用する内部変数をすべて初期化します．初期化される内容は「マウス機能の初期化（機能番号0）」で初期化される内容と同じです．初期化される内容については表5.3の初期化される内部変数を参照してください．

MOUSE 0,33

[戻り値]

AX =　　　　−1　正常に初期化した

　　　　　　33　マウスドライバが組み込まれてない

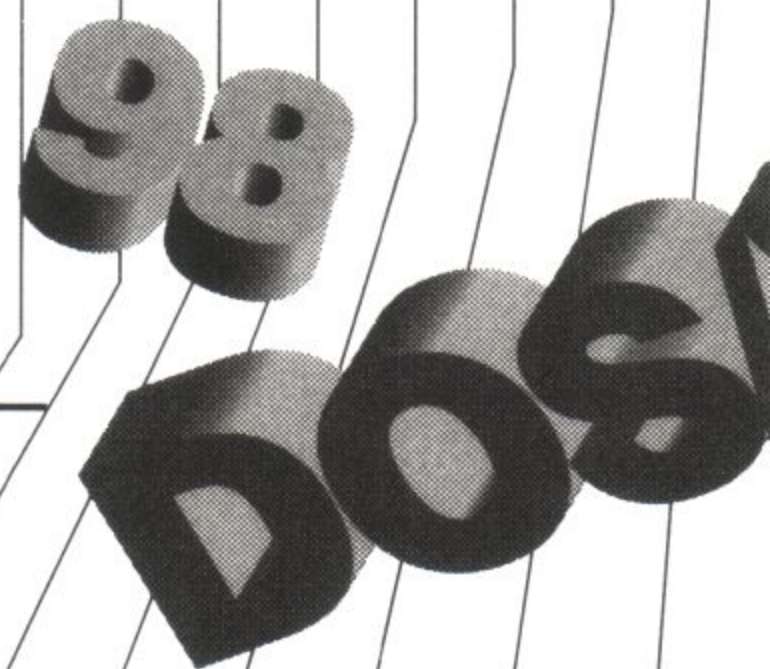
BX =　　　　AXが−1のときは2が返る

リスト5.26　マウスドライバのソフトウェアリセット

```
【Cからの呼び出し】

    SoftResetMouse() ;


【MASM6ルーチン】

;==========================================================================
;       Description : マウスドライバのソフトウェアリセット
;       Sequence    : int   SoftResetMouse(void) ;
;       Paramater   : void
;       戻り値      : 0: マウスドライバが使用可能
;                     -1: マウスドライバが使用不可
;==========================================================================
SoftResetMouse      proc
                    MOUSE   0,33
                    ret
SoftResetMouse      endp
```

第6章

98からDOS/Vの世界へ

DOS/V

タイマ編

この章では，標準タイマICの8253を使用した割り込みプログラミングおよびタイマBIOSについて解説します．また，タイマICを使用したサウンドの発生方法に関しても解説します．

6-1 基礎知識

IBM PC系機種のタイマLSIとしては，PC-9801などと同様にプログラマブルインターバルタイマ8253系（通常は8254）のチップが使用されています．8253はプログラマブルなインターバルタイマ／カウンタで，独立した3つの16ビットカウンタを内蔵しています．カウンタの動作はBCDでもバイナリでも動作可能で，それぞれのカウンタが6つの動作モードを選択できるようになっています．しかし，8253のもつ3個のカウンタ（カウンタ0からカウンタ2）はすべて使用されていて，ユーザーには解放されていません．また，システムの基本的なハードウェアやBIOS機能もCPUの種類やクロック周波数に依存しない定期的な間隔を必要とするため，カウンタ0/1を変更することは基本的にできません．また，カウンタ2は，スピーカから発生するの音の周波数を決めるために使用されています．

8253のレジスタは，表6.1のように割り当てられています．

制御レジスタの各ビットは次の意味をもっています．

◎ビット7-6　SC1,SC2(Select Counter)

8253には16ビットのカウンタが3個内蔵されており，そのうちの1つを指定します．

◎ビット5-4：RL1,RL0(Read/Load)

カウンタに対しては8ビット単位で入出力を行います．その方法を指定します．ここで指定されたカウントダウン値が0になった場合に割り込みが発生します．また，最大値は0が設定された場合で，65536（2の16乗）と同じ意味になります．

表6.1　8253のレジスタ　　8253の技術資料

レジスタ	ポートアドレス	用途		
カウンタ0	40h	インターバルタイマ(約18.2Hz)		
カウンタ1	41h	DRAMリフレッシュ信号発生用		
カウンタ2	42h	スピーカー用		
制御	43h	ビット7-6	カウンタ選択	00=カウンタ0
				01=カウンタ1
				10=カウンタ2
		ビット5-4	読み出し指定	00=カウンタラッチ
				01=LSBの入出力
				10=MSBの入出力
				11=LSB,MSBの順に入出力
		ビット3-1	モード指定	000=モード0（単発）
				001=モード1
				x10=モード2
				x11=モード3（方形波）
				100=モード4
				101=モード5
		ビット0	カウンタタイプ	0=バイナリ
				1=BCD

◎ビット3-1：M2,M1,M0(Mode)

割り込みをかけるタイプを指定します．8253は6種類のモードをもっていますが，通常は，モード0（カウント終了時に割り込み）かモード3（方形波レートジェネレータ）を使用します．前者がカウント終了時に一度だけ割り込みが起こるのに対して，後者はカウンタ周期ごとに割り込みが発生します．

◎ビット0　：BCD

BCDカウントかバイナリカウントかを指定します．

すべてのIBM PC系機種では8253のカウントクロック入力は1.19318MHzになっています．ROM BIOS初期設定時にカウンタ0のカウントダウン値0，割り込みモード3として設定されます．したがって，カウンタ0は1秒間に1193180/65536回（約18.2回），すなわち約55ミリ秒に1度の割合でタイマ割り込みを定期的に発生させます．この割り込みは，マスタ割り込みコントローラ8259AのIRQ0(INT 08h)に接続されています．

このハードウェアタイマ割り込みハンドラは，後述のシステムタイマのカウントアップ，ディスケットのモータの停止，および割り込み処理内部でINT 1Ch（ユーザー用タイマ刻み）の呼び出しを実行しています．ディスケットのモータ停止は，BIOSワークエリアの40h:40hにあるカウンタをカウントダウンし，0になったところでモータをオフにします．また，INT 1Chは定期的に呼び出されるため，定期的に動作する常駐プログラムなどは通常INT 1Chの割り込みをフックして処理を行っています．この割り込みに関しては，複数のプログラムがフックする可能性がありますので，INT 1Chをフックして使用する場合には，必ず，フック前のアドレスに制御を渡すようにしてください．

一般的なタイマ割り込みの基本ルーチンは，リスト6.1のような形式をとります．

リスト6.1　タイマ割り込みの使用

```
                .code

;****************************************************************************
;*                                                                          *
;*          タイマ割り込みハンドラの設定                                    *
;*          void        SetTimerHandler(void);                              *
;*          戻り値：    なし                                                *
;*                                                                          *
;****************************************************************************
OrgTimerHandler     dword   ?

SetTimerHandler     proc    uses bx dx es
                    MSDOS   351Ch                   ;前のハンドラアドレスの保存
                    mov     word ptr cs:OrgTimerHandler+2, es
                    mov     word ptr cs:OrgTimerHandler, bx
                    MOVSEG  ds, cs
                    mov     dx,offset cs:TimerHandler
                    MSDOS   251Ch                   ;新しいハンドラアドレス
                    ret
SetTimerHandler     endp

;****************************************************************************
;*                                                                          *
;*          タイマ割り込みハンドラの解除                                    *
;*          int         SetTimerHandler(void);                              *
;*          戻り値：    0       ：解除できない                              *
;*                      0以外 ：解除できた                                  *
;*                                                                          *
;****************************************************************************
ResetTimerHandler   proc    uses dx ds
                    lds     dx, cs:OrgTimerHandler  ;元のハンドラアドレス
                    .if dx != 0                     ;元のアドレスが0でないとき
                        MSDOS   251Ch               ;アドレスを元に戻す
                    .else
                        xor     ax, ax              ;元のアドレスが0のとき
                    .endif
                    ret
ResetTimerHandler   endp

;****************************************************************************
;*                                                                          *
;*          タイマ割り込みハンドラ(内部関数)                                *
;*                                                                          *
;****************************************************************************
TimerHandler        proc    private

;                    ：  ここにタイマ割り込みの処理を入れます

                    iret
TimerHandler        endp
```

6-2 55ミリ秒より短い間隔の割り込みが必要な場合

IBM PC系では，55ミリ秒周期のタイマ割り込みしか使用できませんが，それよりも短い周期のタイマ割り込みが必要な場合のプログラミングに関して解説します．

◎タイマ割り込みには，8253カウンタ0しか使用できません．

◎割り込み間隔は，元の割り込み周期の2の整数乗分の1にしか設定できません．

◎元のタイマ割り込みハンドラは，55ミリ秒周期で発生することを前提に作成されていますので，必ず55ミリ秒周期で呼び出せるようにしなければいけません．

◎カウンタ0を変更できるのは，1システムで1プログラムだけです．したがって，常駐プログラムで使用する場合には注意してください．

前述のように，IBM PC系機種では，約55ミリ秒間隔で発生するINT 08hをもとに各種BIOS機能を実行しています．また，大半の常駐プログラムなども，INT 08hから派生するINT　1Chの定期的な間隔を前提にプログラミングされています．したがって，カウンタ0を変更すると，システムの時刻がおかしくなったり，ケースによってはハングアップする可能性もあります．

しかし，ケースによっては55ミリ秒よりも短い周期が必要となることがあります．こういった場合にはINT 08hを自分でフックし，カウンタを元の値（65536）の1/N（Nは2の整数乗）にして，タイマ割り込みのN回目ごとに，元の割り込みハンドラ（INT　08h）を呼び出せばよいわけです．すなわち，元の割り込みハンドラから見れば，55ミリ秒ごとにしかタイマ割り込みが発生していないように見えるからです．また，N－1回は自分で8259AへEOIコマンドを転送して，割り込みを終了させてください．

リスト6.2の関数では，分周の値をパラメータで渡して，割り込み回数を元の周期の分周倍に変更しています．しかし，もともとの周期が65536=10000hをもとにしているため，パラメータの値は必ず2の整数乗（たとえば，2,4,8……）にしてください．途中の値でも動作はすると思いますが，誤差が生じるため，システムタイマが狂ったり，場合によっては動作待ち時間が狂うため，ディスケットなどが読めなくなる場合もあります．

また，メイン関数のほうでは，ltickという名前のunsigned long変数を用意してください．タイマ割り込みが発生するごとにカウントアップされていきます．さらに有効利用したい場合には，このサンプルソースに含まれているRealTimeHandlerを修正するか，またはCのinterrupt関数として再作成してください．ただし，ハードウェア割り込み中ですので，十分プログラミングには注意しなければなりません．

さらに注意しなければならないことは，このテクニックが使用できるのはそのシステムで1つのプログラムだけです．これは，55ミリ秒に1回割り込みが起きることは常識となっ

リスト6.2 55ミリ秒より短い間隔の割り込みREALTIME.ASM

```
include             std.inc

CTL8259             equ     20h                 ;マスタ8259A 制御ポート
DTA8259             equ     02h                 ;マスタ8259A データポート
CTL8253             equ     43h                 ; 8253制御ポート
DTA82530            equ     40h                 ; 8253データポート 0
TIMER_MASK          equ     01h                 ;マスタIRQ0 割り込みマスク
EOI                 equ     20h                 ;

                    .data
                    extrn   ltick:dword         ;この関数を組み込むメインルー
                                                ;チンで使用されるカウンタ

                    .code

;***************************************************************************
;*                                                                         *
;*          タイマ割り込みハンドラの設定                                   *
;*          void        StartRealTime(uint cycle);                         *
;*          パラメータ：cycle    分周                                      *
;*          戻り値：    なし                                               *
;*                                                                         *
;***************************************************************************
OrgRealTimeHandler  dword   ?

StartRealTime       proc    uses bx cx dx ds es, cycle:word
                    cli                         ;割り込み禁止
                    in      al, DTA8259
                    or      al, TIMER_MASK
                    out     DTA8259, al         ;タイマ割り込みマスクをセット
                    sti                         ;割り込み解除

                    MSDOS   3508h               ;前のハンドラアドレスの保存
                    mov     word ptr cs:OrgRealTimeHandler+2, es
                    mov     word ptr cs:OrgRealTimeHandler, bx
                    MOVSEG  ds, cs
                    mov     dx,offset cs:RealTimeHandler
                    MSDOS   2508h               ;新しいハンドラアドレス

                    mov     al, 36h             ;方形波(モード3)
                    out     CTL8253, al
                    mov     cx, cycle           ;分周
                    mov     cs:ncount, cl       ;分周の保存
                    xor     ax, ax
                    mov     cs:counter, al      ;分周カウンターの初期化
                    mov     dx, 1
                    div     cx
                    out     DTA82530, al
                    mov     al, ah
                    out     DTA82530, al        ;カウンターに割り込み周期をセット

                    cli                         ;割り込み禁止
                    in      al, DTA8259
                    and     al, not TIMER_MASK
                    out     DTA8259, al         ;タイマ割り込みマスク解除
                    sti                         ;割り込み解除
                    ret
StartRealTime       endp

;***************************************************************************
;*                                                                         *
;*          タイマ割り込みハンドラの解除                                   *
;*          int         StopRealTime(void);                                *
;*          戻り値：    0       ：解除できない                             *
;*                      0以外   ：解除できた                               *
;*                                                                         *
;***************************************************************************
StopRealTime        proc    uses dx ds
                    cli                         ;割り込み禁止
                    in      al, DTA8259
                    or      al, TIMER_MASK
                    out     DTA8259, al         ;タイマ割り込みマスクをセット
                    sti                         ;割り込み解除

                    lds     dx, cs:OrgRealTimeHandler  ;元のハンドラアドレス
                    .if dx != 0                     ;元のアドレスが0でないとき
                        MSDOS   2508h               ;アドレスを元に戻す
                    .else
                        xor     ax, ax              ;元のアドレスが0のとき
                    .endif

                    mov     al, 36h             ;方形波(モード3)
                    out     CTL8253, al
                    xor     ax, ax              ;元の分周
                    out     DTA82530, al
                    mov     al, ah
                    out     DTA82530, al        ;カウンタを元の値に戻す

                    mov     word ptr cs:OrgRealTimeHandler+2, ax
                    mov     word ptr cs:OrgRealTimeHandler, ax

                    cli                         ;割り込み禁止
                    in      al, DTA8259
                    and     al, not TIMER_MASK
                    out     DTA8259, al         ;タイマ割り込みマスク解除
                    sti                         ;割り込み解除
                    ret
StopRealTime        endp

;***************************************************************************
;*                                                                         *
;*          タイマ割り込みハンドラ(内部関数)                               *
;*                                                                         *
;***************************************************************************
ncount              db      ?                   ;分周
counter             db      ?

RealTimeHandler     proc    private
                    pushf
                    push    ax

                    push    ds
                    mov     ax,@data
                    mov     ds, ax
                    add     word ptr ltick, 1
                    adc     word ptr ltick+2, 0
                    pop     ds

                    inc     cs:counter          ;割り込み回数カウントアップ
                    mov     al, cs:counter
                    .if al == cs:ncount         ; N回目のとき
                        xor     al, al
                        mov     cs:counter, al  ;分周カウンタの初期化
                        pop     ax              ;レジスタの復帰
                        popf
                        jmp     cs:OrgRealTimehandler
                                                ;元の割り込みハンドラを呼び出す
                    .endif
                                                ; N回目以外のとき
                    cli                         ;割り込みを禁止する
                    mov     al, EOI
                    out     CTL8259, al         ; 8259Aへ EOI 転送
                    pop     ax
                    popf
                    iret
RealTimeHandler     endp

                    end
```

ており，他のプログラムがINT 08hをフックしてしまえば，そのプログラムは55ミリ秒で割り込みが発生することを前提に書かれているわけですから，動作がおかしくなっても何もいえません．

この関数は，大きく以下の3つに別れています．

タイマ割り込みルーチンの設定（StartRealTime）

◎8259Aのタイマ割り込みをマスクする．

◎元のタイマ割り込みベクタ（INT 08h）の内容を取得し，新しいタイマ割り込みハンドラのアドレスを，割り込みベクタにセットする．

◎8253を新周期のモード3に設定.

カウンタ0をモード3にするためには，制御レジスタに36hを書き込み，LSB（下位8ビット），MSB（上位8ビット）の順に分周カウンタに，65536を渡されたパラメータで割った値を書き込みます.

◎8259Aのタイマ割り込みのマスクを解除する.

タイマ割り込みハンドラでの処理（RealTimeHandler）

◎目的の処理を行う（この関数では何もしていない）

◎割り込み回数をカウントする.

このプログラムでは2種類のカウントをしています．ひとつはメインルーチンに渡すために継続してカウントアップしているもので，もうひとつは元のINT 08hを呼び出すタイミングを計算する割り込みカウントです．割り込みカウントがNになった場合には，割り込みカウントを0に戻してから，元のINT 08h割り込みハンドラを呼び出します．この場合は，元のハンドラが8259Aに対してEOIコマンドを発行します．それ以外の場合は，自分で割り込みを終了させる必要性があるため，8259AへEOIコマンドを転送して，終了します.

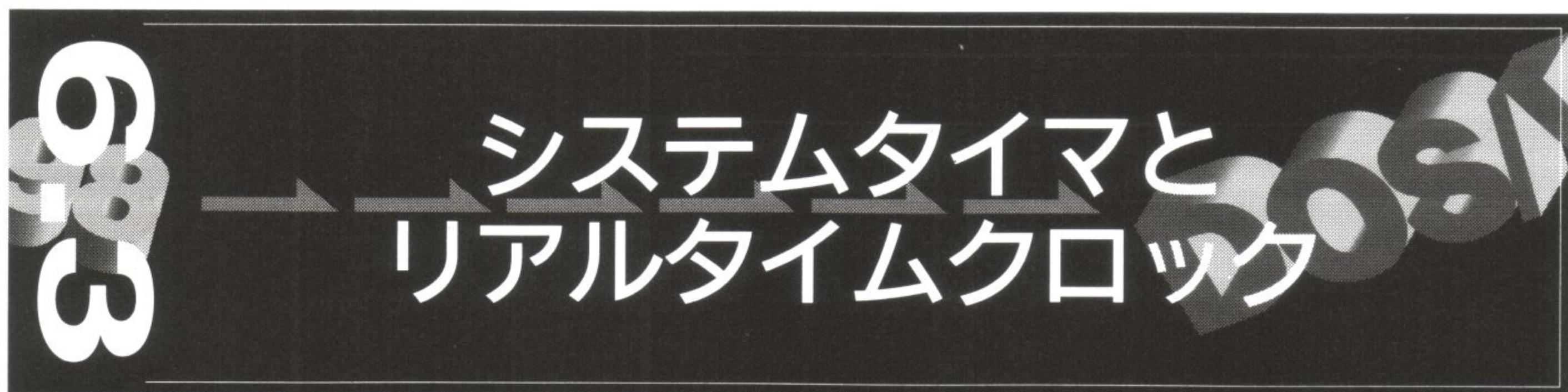
タイマ割り込みルーチンの解除（StopRealTimer）

◎8259Aのタイマ割り込みをマスクする

◎タイマ割り込みベクタの内容を元の値に戻す.

◎8253を元の周期のモード3に設定し，分周カウンタには0（65536と同じ意味）を書き込む.

◎8259Aのタイマ割り込みのマスクを解除する.

6-3 システムタイマとリアルタイムクロック

AT互換機以降の機種では，バッテリバックアップされたリアルタイムクロックが追加されています．このチップとしては，モトローラのMC146818Aが使われており，大きく3つの機能をもっています.

◎PC本体の電源を落としても，ほぼ正確な日付・時刻を維持すること.

◎一定周期ごとに割り込みを発生させる.（INT 70h）

◎システム構成をCMOS-RAM（64バイト）に記憶させる.

リスト6.3　タイマウェイト関数TMRWAIT.ASM

```
include         std.inc

bios            segment at 0
                org     046Ch
SystemTimer     word    ?                          ;システムタイマ
                                                   ;本当は，dword

                .code

;*****************************************************************************
;*                                                                           *
;*          指定時間待機関数                                                 *
;*          void        TimerWait(uint count);                               *
;*          パラメータ：count   待機時間（1 = 約55ms）                       *
;*          戻り値：    なし                                                 *
;*                                                                           *
;*****************************************************************************
TimerWait       proc    uses cx, count:word
                mov     cx, count
                .if cx != 0                        ;カウンタ=0のときは処理しない
                    xor     ax, ax
                    mov     es, ax
                    assume  es:bios
                    mov     di, offset SystemTimer ;システムタイマ
                    .repeat
                        mov     ax, es:[di]        ;現在の時刻
                        .repeat
                        .until ax != es:[di]       ;タイマがカウントアップ
                                                   ;されなければ継続
                    .untilcxz                      ;カウンタが0になるまで
                    assume  es:nothing
                .endif
                ret
TimerWait       endp

;*****************************************************************************
;*                                                                           *
;*          指定時間待機関数                                                 *
;*          void        TimerWaitSys(uint count);                            *
;*          パラメータ：count   待機時間（1 = 約1ms）                        *
;*          戻り値：    なし                                                 *
;*                                                                           *
;*****************************************************************************
TimerWaitSys    proc    uses cx dx, count:word
                mov     ax, count
                .if ax != 0                        ;カウンタ=0のときは処理しない
                    mov     cx, 1000               ; 1msを 1000μs に変換
                    mul     cx
                    mov     cx, dx                 ; cx:dxに μs単位で設定
                    mov     dx, ax
                    INT15   86h                    ;待ち時間経過待ち
                .endif
                ret
TimerWaitSys    endp

                end
```

日付・時刻に関しては，IPL時にリアルタイムクロックから読み込み，あとは上記カウンタ0を使用して時刻を刻んでいます．この時刻（実際には0時を0とするカウンタであり，1秒間に約18.2ずつカウントアップされ，24時間で1800B0hになります．したがって，時刻の形式にはなっておらず，DOSで読み出すときに時刻形式に変換されます）は，以下のBIOSデータエリアに書き込まれています．

アドレス	意味	サイズ
0040h:006Ch	システムタイマカウンタ	2ワード
0040h:0070h	システムタイマオーバーフロー	1バイト

システムタイマを使用した，タイマウェイト関数をリスト6.3にのせておきます．本来は以下のタイマBIOSを使用してシステムタイマを読み出すことができるのですが，今回は直接BIOSワークエリアを見る形式にしています．また，このシステムタイマは，約55ミリ秒に1ずつカウントアップされますので，この関数では55ミリ秒の整数倍の時間だけウェイトすることができます．

この日付・時刻を読み取り・設定するために，タイマBIOS（INT 1Ah）が用意されています．ただし，設定に関しては，システムタイマとリアルタイムクロックを同時に設定しなければならないため，DOSのシステムコールを使用するように『IBM DOS/Vバージョン J5.0/V BIOSインターフェース技術解説書』に書かれています．

日付・時刻の読み取りに関しては，以下のタイマBIOSを使用します．

システムタイマの時刻カウントの読み取り

TIMER　00h

[戻り値]

CX =	カウントの高位部	
DX =	カウントの低位部	
AL =	0	タイマが前回読み出してから，24時（午前0時）を経過していない

0以外　24時を経過した

このシステムタイマカウントと24時の経過の判断は，上記BIOSワークエリアから取得されます．また，DOSで読み出される時刻も，このカウントから計算されており，リアルタイムクロックの値は使用されていません．このシステムタイマカウントは，約55ミリ秒ごとに1ずつカウントアップされています．

リアルタイムクロックの時刻の読み取り

TIMER　02h

[戻り値]

CH =	時（BCD形式）	
CL =	分（BCD形式）	
DH =	秒（BCD形式）	
DL =	0	標準時間制
	1	夏時間制
CF =	0	クロックは作動
	1	クロックが停止しているか時刻の更新中

この値は，リアルタイムクロックのCMOS-RAMから読み取られます．また，内部的に時刻を更新している場合に，キャリーフラグがオンで返ってくる場合がありますので，その場合には再度，読み取って下さい．

リアルタイムクロックの日付の読み取り

TIMER　04h

[戻り値]

CH =	世紀（BCD形式）	
CL =	年（BCD形式）	
DH =	月（BCD形式）	
DL =	日（BCD形式）	
CF =	0	正常終了
	1	エラー

この値は，リアルタイムクロックのCMOS-RAMから読み取られます．

日付・時刻の設定には，前述のようにタイマBIOSではなく，以下のDOSのシステムコールを使用します．

システム日付の設定

CX =	年（1980-2099）
DH =	月
DL =	日

MSDOS 2Bh

[戻り値]

AL =	0	正常終了
	FFh	日付がおかしい．システム日付は更新されません．

システム時刻の設定 ――――――――――――――――――――――

CH =　　　時

CL =　　　分

DH =　　　秒

DL =　　　1/100秒（実際には，約55ミリ秒ごとにしか更新されないので意味は薄い）

MSDOS 2Dh

[戻り値]

AL =　　　0　　正常終了

　　　　　FFh　時刻がおかしい．システム時刻は更新されません．

この両者のシステムコールでは，BIOSワークエリアにあるシステムタイマとリアルタイムクロックの両者が更新されます．

また，8254とは別に，リアルタイムクロックも一定周期で割り込みを発生させています．リアルタイムクロックのほうはINT 70h(IRQ 8)に接続されており，1/1024秒に1回割り込みが起きます．ROM BIOSは，システムBIOS(INT 15h)のうち，イベントセット（AH=83h），時間待ち（AH=86h）で使用されるカウンタを更新します．また，以下のタイマBIOSで指定されたアラーム時刻になった場合には，INT 4Ahを呼び出します．

リアルタイムクロックのアラーム設定 ――――――――――――

CH =　　　時（BCD形式）

CL =　　　分（BCD形式）

DH =　　　秒（BCD形式）

TIMER　06h

[戻り値]

CF =　　　0　　正常終了

　　　　　1　　すでににアラームがセットされているか，クロックが停止中

リアルタイムクロックのアラーム解除 ――――――――――――

TIMER　07h

[戻り値]

なし

このアラーム処理は，システムで1つしか登録できません．したがって，まず，INT 4Ahが指すアドレスの値が，IRETかどうかを判定して，IRETの場合にのみ，割り込みをフックすることができます．

CMOS-RAMは64バイトあり，このCMOS-RAMもバッテリバックアップされているため，システムのハードウェア構成の情報などが記録されています（これがよくいわれるCMOSです）．

CMOSの割り当ては，表6.2のようになっています．

表6.2 CMOS-RAMの構成

◎I/Oポート

アドレス	入出力	名　称	ビット	内　容
70h	R/W	リアルタイム/CMOS,NMIマスク	7	NMI割り込みの許可
			6	予約済み
			5-0	CMOS-RAMのアドレス
71h	R/W	指定アドレスのCMOS-RAMの値		

◎リアルタイムクロック　カレンダ関連

CMOS-RAM アドレス	入出力	ビット	内容
00h	R/W	7-0	秒 (BCD形式)
01h	R/W	7-0	アラーム秒 (BCD形式)
02h	R/W	7-0	分 (BCD形式)
03h	R/W	7-0	アラーム分 (BCD形式)
04h	R/W	7-0	時 (BCD形式)
05h	R/W	7-0	アラーム時 (BCD形式)
06h	R/W	7-0	曜日 (BCD形式)
07h	R/W	7-0	日 (BCD形式)
08h	R/W	7-0	月 (BCD形式)
09h	R/W	7-0	年 (BCD形式)

◎リアルタイムクロック　ステータス関連

CMOS-RAM アドレス	入出力	ビット	内容
0Ah	R/W	7	1 = RTC時刻更新中
		6-4	クロック分周比
		3-0	出力周波数/割り込み周期
0Bh	R/W	7	RTC 0 = 動作中,1 = 停止中
		6	1 = RTC周期割り込み許可
		5	1 = アラーム割り込み許可
		4	1 = 時刻更新割り込み許可
		3	1 = 方形波割り込み許可
		2	時刻・日付の形式　0 = BCD　,1 = バイナリ
		1	時刻の表記　0 = 12時間,1 = 24時間
		0	1 = 夏時間の指定
0Ch	R		(一度読み出すと,消去される)
		7	IRQ要求フラグ
		6	周期割り込みフラグ
		5	アラーム割り込みフラグ
		4	時刻更新割り込みフラグ
		3-0	予約済み
0Dh	R		(一度読み出すと,消去される)
		7	1 = RTC動作中
		6-0	予約済み

◎リアルタイムクロック　診断コード

CMOS-RAM アドレス	入出力	ビット	内容
0Eh	R/W	7	RTCバッテリ切れ
		6	CMOS-RAM チェックサムエラー
		5	セルフテスト　構成エラー
		4	セルフテスト　メモリサイズエラー
		3	ハードディスク初期化エラー
		2	RTC時刻がおかしい
		1-0	予約済み

◎リセット要因

CMOS-RAM アドレス	入出力	ビット	内容
0Fh	R/W		0 : パワーオンリセット
			1 : リアルモードへの移行
			4 : セルフテスト終了　システムのブート
			5 : EOI発行後,40h:67hの指すアドレスへジャンプ
			8 : メモリセルフテストへの復帰

9 : システムBIOS (INT 15h), AH=87h機能への復帰
10: EOIなしで,40h:67hの指すアドレスへジャンプ
11: 40h:67hの指すアドレスへIRET
12: 40h:67hの指すアドレスへRET

◎システム構成

CMOS-RAM アドレス	入出力	ビット	内容
10h	R/W	7-4	ディスケットドライブ0のタイプ
		3-0	ディスケットドライブ1のタイプ
			0000B : ドライブなし
			0001B : 360KB
			0010B : 1.2MB
			0011B : 720KB
			0100B : 1.4MB
11h	---		予約済み
12h	R/W	7-4	ハードディスク0のタイプ
		3-0	ハードディスク1のタイプ
13h	---		予約済み
14h	R/W	7-6	ディスケットドライブの個数
			00B = 1個, 01B = 2個
		5-4	プライマリディスプレータイプ
			00B = C000h:0hにROMがあるカード
			01B = 40桁カラー
			10B = 80桁カラー
			11B = モノクローム
		3-2	予約済み
		1	数値演算コプロセッサ搭載
		0	ディスケットからブート可能
15h-16h	R/W	15-0	DOSの基本メモリ容量 (1KB単位)
17h-18h	R/W	15-0	拡張メモリ容量 (1KB単位)
19h	R/W		12hの値がFxhのときの, ハードディスク#0拡張タイプ
1Ah	R/W		12hの値がxFhのときの, ハードディスク#1拡張タイプ
1Bh-2Dh	---		予約済み
2Eh-2Fh	R		CMOS-RAMの10h～2Dhまでのチェックサム
30h-31h	R/W		拡張メモリ容量 (1KB単位)
32h	R/W		世紀 (BCD形式)
33h	---		情報フラグ (IPL中にセットされる)
34h-3Fh	---		予約済み

6-4 サウンド

ここでは, タイマチャンネル2を利用したPCスピーカの操作について述べていくことにします.

◎音程はタイマチップのカウンタ2の値で決まります.

◎スピーカのオン／オフは, システム制御ポートBの下位2ビットを使用します.

通常, 人が聞き取れる「音」の範囲は20～2万Hzくらいといわれており, 人の話す言葉(音) は約100～1000Hzとなっています. IBM PCでは, 18～100万Hzまでの範囲の「音」をスピーカから発生させることが可能で, この範囲の音声を発生させるためのタイマチップ (8253) のチャンネル2を専用でもっています.

さて，このタイマチップはチャンネル2用の1ワードのカウンタをもっており，このカウンタは1193180Hzのクロックサイクルで減算していて，カウンタが0になるごとに音声発生回路へパルスを送り，カウンタ値を元の値に戻して，これを繰り返しています．つまり，タイマは出力したい周波数を出すために，クロック周波数をカウント値で割り，特定の周波数の音声発生用の一連のパルスを送り出しているのです．

ですから，欲しい周波数に必要なカウントを計算して，それをタイマチップのカウンタ2に設定すればよいわけです．

カウントの算出法としては“COUNT＝1193180/周波数”で求めることができます．この式からわかるとおり，“周波数 = 1193180/COUNT”ですから，1193180を1ワードの最大値65535で割った18が周波数の最小値であり，1ワード最少の1で割った1193180が周波数の最大値となります．

上記の式で求めたカウンタ値は，I/Oポートを通してタイマチップのカウンタ2にセットします．これは図6.1の手順でタイマをレディ状態にセットしてから，カウンタ値を下位1バイト，上位1バイトの順で送り出します．

さて図6.1の手順で，8253へカウンタ2の値を設定しました．ですが，これだけではまだ音は出ません．次にスピーカを音が出るような状態にしてやらなければなりません．スピーカの制御はI/Oポート61H（システム制御ポートB）の下位2ビットで制御します．下位2ビット以外のビットは図6.2に示すとおり，他の目的で使用されているため，直接このポートの値を書き換えてはいけません．

リスト6.4に，スピーカのオン／オフの方法を示します．スピーカはオンにすると，次にオフになるまで音をならし続けます．プログラムを作成する人は，このオン／オフの間隔を適度な長さに調節しなければなりません．この間隔の長さだけでも，同じ周波数なのに別な音に聞こえることもあります．また，スピーカの音の大きさを調節することはできませんが，周波数によって音の大きさはかなり変化するということに留意してください．

図6.1

タイマチップへのカウンタ送出方法

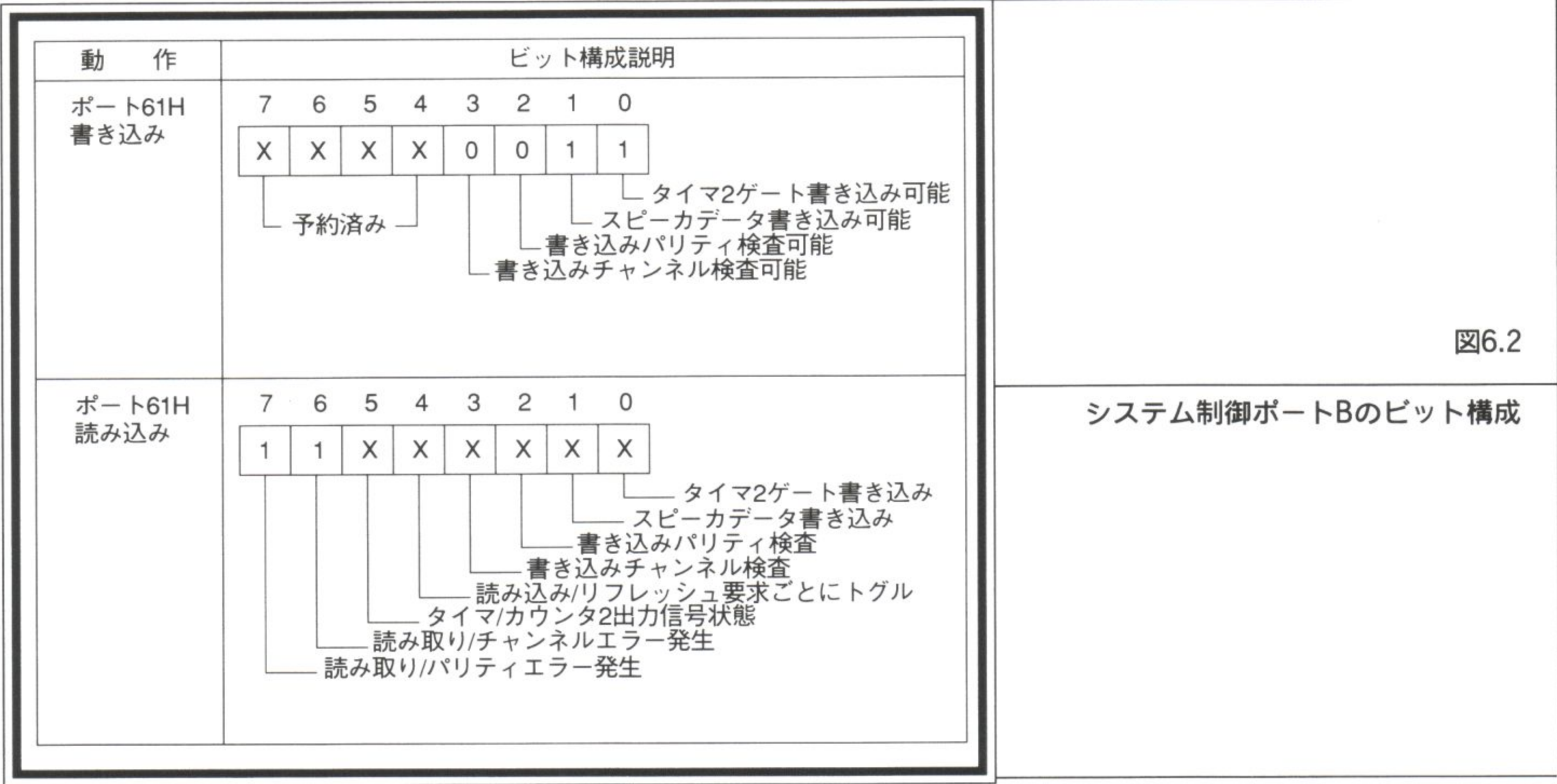
	ポート	送出内容	意　味
手順1	43H	0B6H	タイマチップをレディ状態にセット ビット 7 6 5 4 3 2 1 0 = 1 0 1 1 0 1 1 0 ビット0：バイナリ16ビットカウンタ ビット3-1：方形波くり返し発生 ビット5-4：LSBをロードして，次にMSBをロード ビット7-6：カウンタ2を選択
手順2	42H	カウンタ下位バイト	カウンタ値の下位1バイトを送出
手順3	42H	カウンタ上位バイト	カウンタ値の上位1バイトを送出

```
mov   al, 0B6h          ; 手順1 タイマチップのレディ
out   43h, al
mov   ax, BuzzerCount   ; 手順2 下位バイトの送出(LOAD LSB)
out   42h, al
xchg  ah, al            ; 手順3 上位バイトの送出(LOAD MSB)
out   42h, al
```

添付ディスクのサンプルプログラムBUZZ.C（リスト6.5）では，リスト6.4の方法で通常の半音階スケールで定義されている周波数の音を数種類発生させています．音の長さはプログラム中とプログラムの冒頭で定義されていて，この間隔の値は55ミリ秒を1という単位で表しています．

動　作	ビット構成説明
ポート61H 書き込み	ビット 7 6 5 4 3 2 1 0 = X X X X 0 0 1 1 7〜4：予約済み 3：書き込みチャンネル検査可能 2：書き込みパリティ検査可能 1：スピーカデータ書き込み可能 0：タイマ2ゲート書き込み可能
ポート61H 読み込み	ビット 7 6 5 4 3 2 1 0 = 1 1 X X X X X X 7：読み取り/パリティエラー発生 6：読み取り/チャンネルエラー発生 5：タイマ/カウンタ2出力信号状態 4：読み込み/リフレッシュ要求ごとにトグル 3：書き込みチャンネル検査 2：書き込みパリティ検査 1：スピーカデータ書き込み 0：タイマ2ゲート書き込み

図6.2

システム制御ポートBのビット構成

リスト6.4　スピーカのオン／オフの方法

```
        in      al,61h
        or      al,3
        out     61h,al              ; スピーカをONにする

             .
             .
             .                      ; 適度にウェイトをかける
             .
             .

        in      al,61h
        and     al,( not 3 )        ; スピーカをOFFにする
        out     61h,al
```

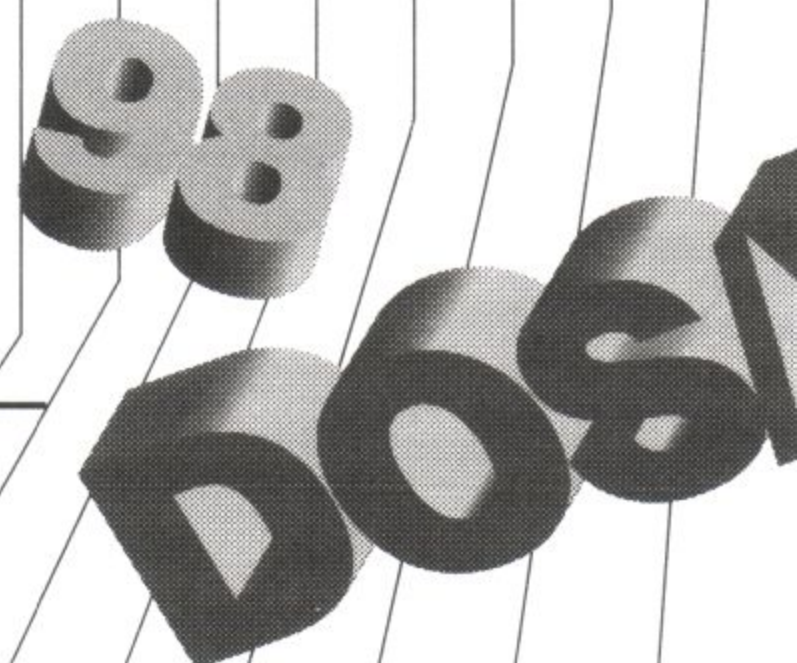
リスト6.5　ブザーを鳴らすBUZZ.C

```
#include        <stdio.h>

#include        "common.h"

int             BuzzWaitValue = 15 ;
uint    BuzzerCount ;

void    Buzzer(int BuzzWaitValue, uint BuzzerCount) ;

double  hz_tbl[] = {
                          65.41 ,  69.30 ,  73.42 ,  77.78 ,  82.41 ,  87.31 ,
                          92.50 ,  98.00 , 103.83 , 110.00 , 116.54 , 123.47 ,
                         130.81 , 138.59 , 146.83 , 155.06 , 164.81 , 174.61 ,
                         185.00 , 196.00 , 207.65 , 220.00 , 233.08 , 246.94 ,
                         261.63 , 277.18 , 293.66 , 311.13 , 329.63 , 349.23 ,
                         369.99 , 392.00 , 415.30 , 440.00 , 466.16 , 493.88 ,
                         523.25 , 554.37 , 587.33 , 622.25 , 659.26 , 698.46 ,
                         739.99 , 783.99 , 830.61 , 880.00 , 932.33 , 987.77 ,
                        1046.50 ,1108.73 ,1174.66 ,1244.51 ,1328.51 ,1396.91 ,
                        1479.98 ,1567.98 ,1661.22 ,1760.00 ,1864.66 ,1975.53 } ;
                /*      C        C#       D        D#       E        F        */
                /*      F#       G        G#       A        A#       B        */

main()
{

        int             i ;
        double  hz ;
```

```
	for ( i = 3 ; i < 53 ; i += 3 ) {
		BuzzerCount = (unsigned)( (double)1193180.0 / hz_tbl[i] ) ;
		printf("Buzzer [%7.2f Hz] = %5u¥n",hz_tbl[i],BuzzerCount) ;
		if ( i > 50 )
			BuzzWaitValue = 75 ;
		Buzzer(BuzzWaitValue,BuzzerCount) ;
	}

	BuzzWaitValue = 15 ;
	for ( i = 52 ; i >= 3 ; i -= 3 ) {
		if ( i == 4 )
			hz = hz_tbl[i+4];
		else
			hz = hz_tbl[i];
		BuzzerCount = (unsigned)( (double)1193180.0 / hz ) ;
		printf("Buzzer [%7.2f Hz] = %5u¥n",hz,BuzzerCount) ;
		if ( i < 6 )
			BuzzWaitValue = 75 ;
		Buzzer(BuzzWaitValue,BuzzerCount) ;
	}
	BuzzerCount = (unsigned)( (double)1193180.0 / hz_tbl[36] ) ;
	BuzzWaitValue = 0;
	Buzzer(BuzzWaitValue,BuzzerCount) ;

}
```

第7章

98
98からDOS/Vの世界へ
DOS/V

98
DOS/V

その他の周辺機器編

　この章では，いままで解説しなかった，プリンタ（パラレルポート）関連とディスク関連のトピックスをいくつか解説します．

7-1 パラレルポート

AT互換機以降の機種では，プリンタポートとしてパラレルポートが使用されています．すなわち，パラレルポートは8ビットのデータを同時にやりとりするためのもので，一般的にプリンタポートとして使用されていますが，入出力の両方向に動作可能ですので，プリンタ以外にも各種周辺機器を接続する場合があります．

◎AT互換機では，最大4つまでのパラレルポートが使用できます．

◎パラレルポートのI/OベースアドレスはBIOSワークエリアに書き込まれています．

◎パラレルポートのI/O空間の連続した3バイトにマップされています．

パラレルポートに関するBIOSは，プリンタBIOSがそのまま出力に使用できます．しかし，パラレル入力には未対応で，かつ割り込みも未使用で，CPUで転送の終了を待っています．

また，BIOSのワークエリアから見ると，4つまでのパラレルポートをサポートしていますが，プリンタBIOSでは，最大3台までのプリンタにしか対応していません（表7.1，7.2）．

プリンタドライバは，IBMプリンタ用（$PRNUSER.SYS）と，ESC/Pプリンタ用

表7.1　パラレルポートのI/Oポートベースアドレス

アドレス	意　味
40h ： 08h	パラレルポート#1・ベースアドレス　(0のときは未接続)
40h ： 0Ah	パラレルポート#2・ベースアドレス　(0のときは未接続)
40h ： 0Ch	パラレルポート#3・ベースアドレス　(0のときは未接続)
40h ： 0Eh	[AT]パラレルポート#4・ベースアドレス　(0のときは未接続)

表7.2　パラレルポートのI/Oレジスタ

アドレス	入出力	用　途
+0	I/O	パラレルポートのデータ
+1	IN	パラレルポートのステータス ビット 7　0 = ビジー ビット 6　1 = 要求した命令に対する応答 ビット 5　1 = 用紙切れまたは自動給紙機構中のジャム ビット 4　1 = オンライン状態 ビット 3　1 = I/Oエラー ビット 2　0 = 割り込み発生 ビット1-0　予約済み
+2	OUT	パラレルポートの制御 ビット7-6　予約済み ビット 5　[AT]予約済み 　[PS/2]入出力方向フラグ　0=出力，1=入力 ビット 4　1 = 割り込み許可 ビット 3　1 = プリンタセレクト ビット 2　1 = プリンタ初期化 ビット 1　1 = 用紙送り ビット 0　1 = ストローブ

アプリケーション
プログラム
INT 17h
ネットワーク
リダイレクタ
NetWareなど
他システムへ
他システムより
INT 17h
プリンタ
ドライバ
INT 17h
ROM BIOS
I/O
プリンタ

図7.1

印刷データの流れ

（$PRNESCP.SYS）の2種類が存在しています．企業ユーザーは別として，通常は後者を使用すると思われますので，$PRNESCP.SYSに関してだけ説明を行います．通常，DOS/V内部では，全角（2バイト）文字はシフトJISコードで扱われているのに対して，ESC/P系日本語プリンタは全角文字をJISコードで取り扱います．そのため，このプリンタドライバは文字コードを出力する際に，出力データ内のエスケープシーケンス文字などを解釈して，内部のシフトJISコードをJISコードに変換します．また，ユーザー定義文字は，$FONT.SYSよりフォントイメージを取得してプリンタに出力しています．

最新のIBM DOS J5.02/Vでは，漢字フォントROMをもっていない英語仕様のプリンタに対しても，$FONT.SYSが管理するフォントを使用して日本語印刷が可能となっています．

しかし，特殊なプリンタや他の装置（プロッタなど）を接続する場合は，データストリームをいじられたくない場合があります．そういった場合には，CONFIG.SYSより$PRNESCP.SYSの指定をはずすようにすれば，完全に透過モードになるので，問題はありません．

また，NEC PC-PR201系用のプリンタドライバは標準では添付されていませんが，ライオスよりPC-PR201用の印刷ドライバが販売されていますので，DOS/VでPC-PR201に印刷することも可能です．

7.2 ドライブ制御

ここでは，アプリケーションでいろいろと利用できそうな便利な関数をいくつか紹介します．ここで紹介する内容は，DOS/Vに依存したもの，MS-DOS汎用のものなどいろいろととりまぜて，ドライブに関連するサンプルを取り扱っていきます．

「ドライブB:にフロッピーを差し込み，キーをどれか押して下さい．」

「おや，どこかで見たメッセージだな？」と思われた方も多いのではないでしょうか．このメッセージは，1ドライブシステムでBドライブにアクセスにいこうとすると，アプリケーションの実行中でも強引に画面に割り込んできて，2行に渡って表示して逃げていきます．このメッセージのために，「INT 29Hをフックして，このメッセージがきたらスキップするようにした」という人もいるくらいですが，そこまでしなくてもこのメッセージを出さなくすることはできるのです．「ドライブ制御」のまずはじめは，この「1ドライブシステム時の“フロッピー交換メッセージ”を出さなくするテクニック」を紹介しましょう．

まずやらなければならないのは，現在使用しているシステムが1ドライブかどうかを調べることです．これはBIOSワークエリアの0040h:0010hを参照します（図7.2）．

リスト7.1のサンプルプログラムでは，SingleDriveFlagという変数に1ドライブシステムであるかどうかの情報を格納しています．この変数は後述のSetDiskette()関数で参照されるグローバル変数になっています．

さて，準備段階はこれでおわりました．アプリケーションは起動されたらすぐに，まず

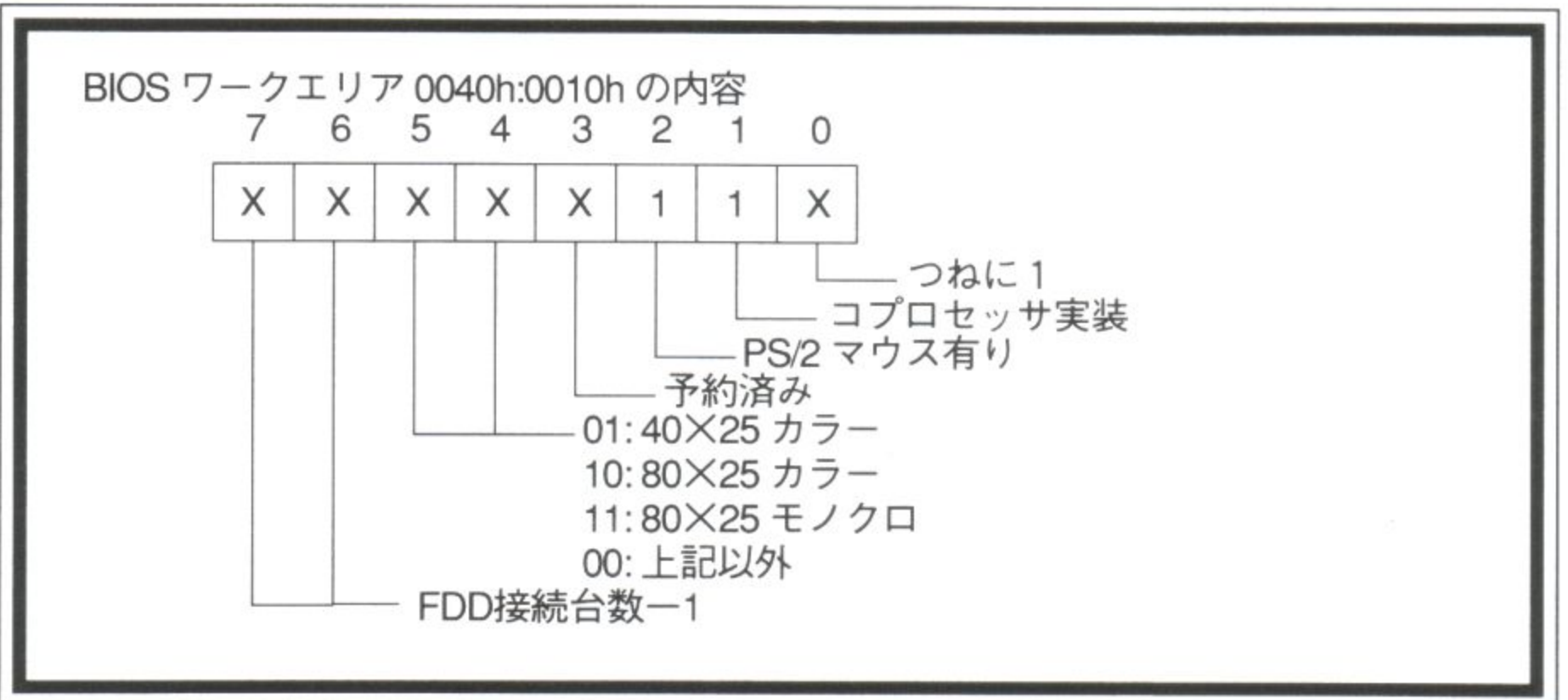
図7.2

1ドライブ検査用BIOS作業エリア

BIOS ワークエリア 0040h:0010h の内容

7	6	5	4	3	2	1	0
X	X	X	X	X	1	1	X

- ビット0: つねに1
- ビット1: コプロセッサ実装
- ビット2: PS/2 マウス有り
- ビット3: 予約済み
- ビット5-4: 01: 40×25 カラー / 10: 80×25 カラー / 11: 80×25 モノクロ / 00: 上記以外
- ビット7-6: FDD接続台数ー1

リスト7.1　1ドライブ検出方法DRIVE.ASM

```
;*****************************************************************************
;*                                                                           *
;*          シングルドライブチェック                                         *
;*          void         CheckSingleDrive(void) ;                            *
;*          パラメータ：なし                                                 *
;*          戻 り 値：なし                                                   *
;*                                                                           *
;*****************************************************************************
CheckSingleDrive    proc    uses bx cx ds si di
            push    ds
            ; BIOS ワークエリア(0040H:0010H)から1バイトロード
            mov     ax, WORK_SEG
            mov     ds, ax
            assume  ds:WORK_SEG
            mov     si, offset ds:IoBlockArea
            lodsb
            pop     ds

            assume  ds:@data

            mov     SingleDriveFlag, 0      ; SingleDriveFlag を OFF

            ; 1ドライブシステムかどうかチェックする
            .if ( al & 01h )
                mov     cl, 6
                shr     al, cl
                .if al == 0
                    inc     al
                    mov     SingleDriveFlag, al ; SingleDriveFlag を ON
                .endif
            .endif

            .if [SingleDriveFlag  != 0
                ; 接続ドライブ管理テーブルを初期化
                mov     cx, 26
                push    ds
                pop     es
                mov     di, offset [DriveConnectTable]
                push    di
                xor     ax, ax
                rep     stosb
                pop     si

                xor     di, di
                mov     cx, 26

                ; 接続ドライブ管理テーブルを設定
                .repeat
                    mov     bx, di          ;ドライブ番号をセット
                    MSDOS   4408h           ; DOS 4408H ﾌｧﾝｸｼｮ発行
                    .if !carry?             ;該当ドライブ接続有り
                        mov     al, 1
                        push    di
                        add     di, si
                        stosb               ;接続ありは 01h セット
                        pop     di
                    .endif
                    inc     di
                .untilcxz
            .endif

            ret
CheckSingleDrive    endp

        end
```

ドライブB:にフロッピーを差し込み， キーをどれか押して下さい．	図7.3 出力されるエラーメッセージ（A）

ドライブA:にフロッピーを差し込み， キーをどれか押して下さい．	図7.4 出力されるエラーメッセージ（B）

CheckSingleDrive()関数を呼んでください．つぎはファイルアクセス，ドライブ変更などを行う場合にアプリケーションでやらなければならない処理を述べます．

たとえば現在1ドライブシステムのマシンを使用しているとして，フロッピードライブはAドライブに割り当てられているとしましょう．ここで，あるアプリケーションからBドライブにアクセスにいったとします．すると当然のことながら例のメッセージが出力されてしまいます．現在選択されているドライブがAなので，メッセージは図7.3のように出ます．

つぎに，再度アプリケーションからAドライブへアクセスにいくと，今度は図7.4と出力されます．

もう大方の読者の方は，このからくりが見えてきたのではないかと思います．そう，DOSは1ドライブ時の「現在選択されているドライブ」をどこかに記憶しているのです．その記憶しているエリアに「現在選択されているドライブはB」という情報が入っているのなら，出力されるメッセージは前述の図7.3の「出力されるエラーメッセージ（A）」が出るし，「現在選択されているドライブはA」という情報が入っているのなら，出力されるメッセージは図7.4の「出力されるエラーメッセージ（B）」が出力されるわけです．

からくりはわかったので，あとはそのエリアがどこにあるのか突き止め，ディスクアクセスを行う前に，アクセス対象のドライブ情報を強引にそのエリアに設定してしまえばいいのです．そうすれば，この邪魔ものはもう二度と画面に表示されなくなるでしょう．

現在選択されているドライブはエリア0050h:0004hに管理されていて，ここにはドライブ番号が0からの相対で格納されています．通常1ドライブ時のフロッピードライブはAまたはBなので，00hまたは01hが格納されることになります．

ライブラリ関数SetDiskette()では，さきにCheckSingleDrive()で接続ドライブをすべて管理していますので，アクセス対象となるドライブが存在すれば，そのドライブ番号をこのエリアに強制的にセットしてしまいます．これはIBM PS/55などの機種で3モードドライブ（仮想ドライブでNEC PC-9801フォーマットも読み書きできるドライブ）が入っているときへの対処です．この3モードの仮想ドライブも，EとFなどと1つのドライブに仮想ドライブ番号を2つ割り当ててしまうからです．このSetDiskette()の方法でエリアの書き換えを行えば，存在しないドライブが現在選択されていることにもならず，さらに「フロッピー差し替えメッセージ」も出力されなくなるというわけです．

図7.5に「カレントドライブ管理エリア」を，そこの書き換えを行っているライブラリ関数SetDiskette()のリストをリスト7.2に掲載します．

図7.5

カレントドライブ管理エリア

リスト7.2

SetDiskette()関数DRIVE.ASM

```
include         std.inc

                .code

;*****************************************************************************
;*                                                                           *
;*      ディスケットセット処理                                                 *
;*      void        SetDiskette(const uchar *AccessFile);                    *
;*      パラメータ：uchar *AccessFile アクセスファイル名                         *
;*      戻り 値：なし                                                          *
;*                                                                           *
;*****************************************************************************
SetDiskette     proc    uses    bx ds si es di , AccessFile:PBYTE
                assume  ds:@data
                .if SingleDriveFlag  != 0       ;単一ドライブのときのみ
                    push    ds
                    if @model gt 3
                    lds     si, AccessFile
                    else
                    mov     si, AccessFile
                    endif
                    lodsb
                    mov     bl,al                ;ファイル名の1文字目
                    lodsb                        ;ファイル名の2文字目
                    pop     ds

                    .if al != ':'                ;ドライブの識別子でないとき
                        MSDOS   19h              ;現在のドライブ名の取得
                        mov     bl, al           ;現在のドライブ名の保存
                    .endif

                ; アクセス先のドライブ番号を取得
                ; ドライブ情報テーブルを検索し, 指定ドライブが接続されていれば
                ; そのドライブ番号をBIOSワークエリアにセットしてしまう.

                    xor     bh, bh
                    or      bl, 20h              ;ドライブ名を小文字化する
                    sub     bl, 61h              ;数字に変換する a=0,b=1...
                    mov     si, offset [DriveConnectTable]

                    .if byte ptr [si+bx  == 01h
                        mov     ax, BIOS_WORK
                        mov     es, ax
                        assume  es:BIOS_WORK
                        mov     di, offset [SelectDrive]
                        mov     al, bl
                        stosb
                    .endif
                .endif

                ret
SetDiskette     endp

                end
```

さて，アプリケーションがCheckSingleDrive()関数を実行すると，つぎからファイルアクセス，ドライブ変更などを行うたびにSetDiskette()関数を呼んで，この関数にアクセスするファイル名（またはアクセスドライブ）を渡してください．これであなたのアプリケーションでは，1ドライブシステムでFDにアクセスしても「ドライブX:にフロッピーを差し込み，キーをどれか押して下さい．」というメッセージは表示されなくなるでしょう．

SetDiskette()の記述は以下の方法で行います．

```
void      SetDiskette(char *AccessName) ;
```

このAccessNameのなかにはアクセスするファイル名“B:¥CONFIG.SYS”や，“C:”などという文字列が指定されます．ファイル名の場合は先頭にドライブ番号が指定されていなければカレントドライブをアクセス対象として，情報の書き換えを行います．

このほか，添付ディスクのライブラリ関数用DRIVE.ASMには「指定ドライブのディスク残容量の取得」や「指定ドライブの接続チェック」などの関数も含まれています．なにかの参考にしていただければ幸いです．

付　録

98からDOS/Vの世界へ

98 DOS/V

資料編

資料編1　BIOS割り込み一覧

資料編2　BIOSワークエリア

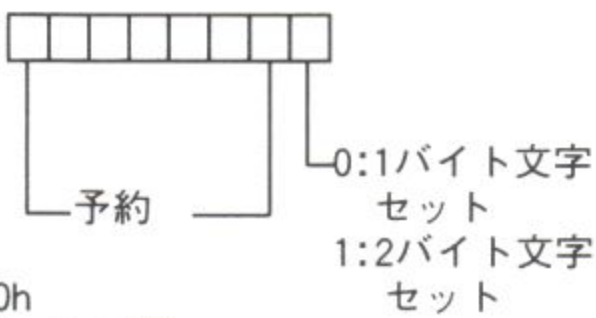
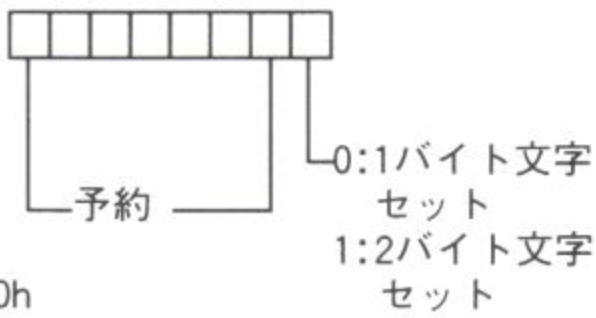
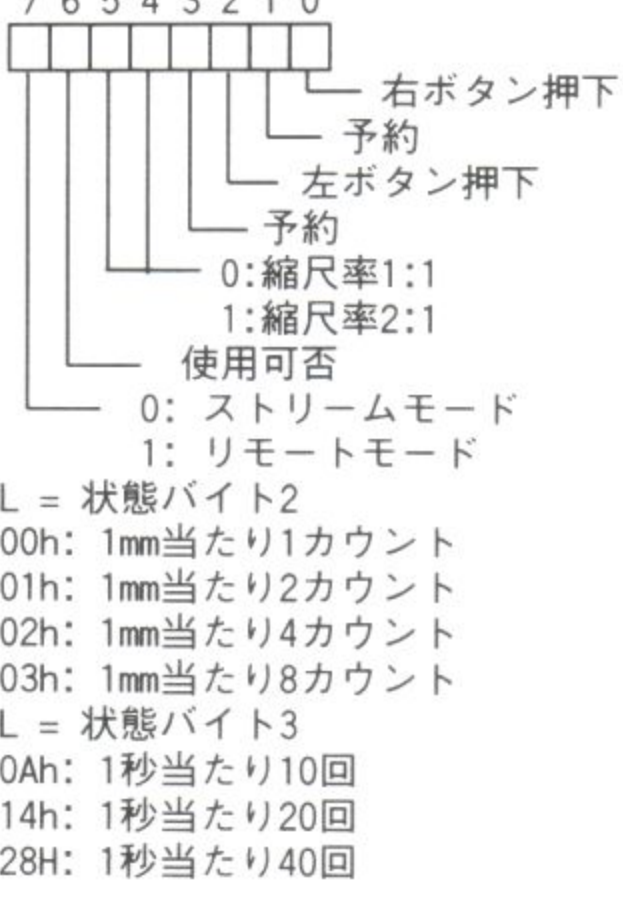
資料編3　I/Oポート

資料編4　割り込み一覧

資料編1 BIOS割り込み一覧

※[PS/2] PS/2対応機種のみ／[Ext] DOS/V Extensionのみ／[Ext/S·D] DOS/V ExtensionとSuper Drivers

割り込み	機能名	機能番号	入力	出力	頁番号
INT 10h	ビデオモードの設定	00h	AH = 00h AL = ビデオモード ビット7 = 1 : VRAMを消去しない		P.31
	カーソル形状の設定	01h	AH = 01h CH = カーソルの開始行 CL = カーソルの終了行 ※ CH = 20h だとカーソル消去		P.38
	カーソル位置の設定	02h	AH = 02h BH = 0 DH = 0からの相対行位置 DL = 0からの相対桁位置		P.38
	カーソル形状・位置の取得	03h	AH = 03h BH = 0	DH = 現在の0からの相対行位置 DL = 現在の0からの相対桁位置 CH,CL = カーソルの形状	P.38
	アクティブページ設定	05h	AH = 05h AL = アクティブページ番号 ※ 通常は 00h を設定		
	画面領域の上方向スクロール	06h	AH = 06h AL = スクロール行数(0 = 矩形消去のみ) BH = 矩形最下行のブランク属性 CH = スクロール開始0からの相対行位置 CL = スクロール開始0からの相対桁位置 DH = スクロール終了0からの相対行位置 DL = スクロール終了0からの相対桁位置		P.49
	画面領域の下方向スクロール	07h	AH = 07h AL = スクロール行数(0 = 矩形消去のみ) BH = 矩形最上行のブランク属性 CH = スクロール開始0からの相対行位置 CL = スクロール開始0からの相対桁位置 DH = スクロール終了0からの相対行位置 DL = スクロール終了0からの相対桁位置		P.49
	現在のカーソル位置の属性と文字の取得	08h	AH = 08h BH = 0	AL = カーソル位置の文字コード AH = カーソル位置の文字属性	P.39
	現在のカーソル位置へ属性と文字を出力	09h	AH = 09h AL = 文字コード BH = 0 BL = 文字属性 CX = 書き込む文字数		P.42
	現在のカーソル位置へ文字を出力	0Ah	AH = 0Ah AL = 文字コード BH = 0 BL = 属性(グラフィック時のみ有効) CX = 書き込む文字数		P.42
	ドットの書き込み(グラフィックモードのみ使用可)	0Ch	AH = 0Ch BH = 0 DX = 0からの相対グラフィック行位置 CX = 0からの相対グラフィック桁位置 AL = カラーコード		
	ドットの読み込み(グラフィックモードのみ使用可)	0Dh	AH = 0Dh BH = 0 DX = 0からの相対グラフィック行位置 CX = 0からの相対グラフィック桁位置	AL = カラーコード	
	テレタイプ式文字と属性の出力	0Eh	AH = 0Eh BL = 文字の色(グラフィック時のみ有効)		P.41

割り込み	機能名	機能番号	入力	出力	頁番号
INT 10h	現在の画面状態の取得	0Fh	AH = 0Fh	AL = 設定されているビデオモード AH = 表示可能1行桁数 BH = アクティブページ(通常は0)	P.31
	パレットレジスタの個別設定	1000h	AH = 10h AL = 00h BL = 設定するパレットレジスタ番号(0-15) BH = カラーレジスタのカラー値		P.52
	オーバースキャンレジスタの設定	1001h	AH = 10h AL = 01h BH = オーバースキャンレジスタに設定するカラーレジスタのカラー値		P.53
	パレットレジスタの一括設定	1002h	AH = 10h AL = 02h ES:DX = レジスタ17バイト (パレットレジスタx16, オーバースキャンレジスタx1)		P.54
	パレットレジスタの個別取得	1007h	AH = 10h AL = 07h BL = 読み取るパレットレジスタの番号	BH = 設定されている カラーレジスタのカラー値	P.52
	オーバースキャンレジスタ取得	1008h	AH = 10h AL = 08h	BH = オーバースキャンレジスタの値	P.53
	パレットレジスター括取得	1009h	AH = 10h AL = 09h ES:DX = レジスタ17バイトのアドレス	ES:DX = 読み取り結果	P.52
	カラーレジスタの個別設定	1010h	AH = 10h AL = 10h BX = カラーレジスタ番号 DH = R 値 CH = G 値 CL = B 値		P.53
	カラーレジスタの一括設定	1012h	AH = 10h AL = 12h BX = 設定する最初のレジスタ番号 CX = 設定カラーレジスタ値の数 ES:DX = カラー値テーブル (R,G,B,R,G,B...)		P.54
	カラーレジスタの個別取得	1015h	AH = 10h AL = 15h BX = 読み取るカラーレジスタ番号	DH = R 値 CH = G 値 CL = B 値	P.53
	カラーレジスタの一括取得	1017h	AH = 10h AL = 17h BX = 読み取る最初のレジスタ番号 CX = 読み取りカラーレジスタ値の数 ES:DX = 読み取り結果を格納する カラー値テーブルのアドレス	ES:DX = 読み取り結果	P.54
	ユーザーフォントの登録	1100h	AH = 11h AL = 00h BH = バイト数 BL = ブロック番号(0に設定) CX = 登録する文字数 DX = 登録する最初の文字コード ES:BP = ユーザーフォントテーブルの アドレス		P.58
	高密度テキストモードへの切り替え [Ext]	1118h	AH = 11h AL = 18h BL = ブロック番号(0に設定)		P.62
	フォントテーブル情報の取得	1130h	AH = 11h AL = 30h BH = 01h	DL = スクロール可能行数-1 CX = 文字フォントの高さ ES:BP = BHで指定されたアドレス	P.65
	DOS/V拡張モードテーブルの取得 [Ext/S・D]	1131h	AH = 11h AL = 31h	ES:BP = 拡張モードテーブルへの ポインタ CX = テーブル項目数	P.63
	DOS/Vフォント品位の切り替え [Ext]	12h	AH = 12h BL = 38h AL = 拡張モードテーブルのｲﾝﾃﾞｯｸｽ番号 または-1(基本テキストモードに戻す) BH = ビデオモード番号(AL=-1の時)	AL = 12h …… 機能が対応 != 12h …… 機能未サポート	P.64
	DOS/Vテキスト密度の切り替え [Ext/S・D]	12h	AH = 12h BL = 39h AL = 拡張モードテーブルのインデックス	AL = 12h …… 機能が対応 != 12h …… 機能未サポート	P.64

割り込み	機　能　名	機能番号	入　　　力	出　　　力	頁番号
INT 10h			番号または-1(モード設定の解除) BH = ビデオモード番号(AL=-1の時)		
	DOS/Vモード設定の取得 [Ext/S·D]	12h	AH = 12h BL = 3Ah	AL = 12h ······ 機能が対応 != 12h ······ 機能未サポート CH = フォント品位設定 CL = テキスト密度設定	P.64
	文字列の書き込み (属性別、カーソル移動なし)	1300h	AH = 13h AL = 00h BH = ページ番号(0を設定) BL = 表示文字列の属性 CX = 書き込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP = 書き込み文字列 ※ 文字列は文字コードを連続して指定		P.42
	文字列の書き込み (属性別、カーソル移動あり)	1301h	AH = 13h AL = 01h BH = ページ番号(0を設定) BL = 表示文字列の属性 CX = 書き込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP = 書き込み文字列 ※ 文字列は文字コードを連続して指定		P.42
	文字・属性列の書き込み (カーソル移動なし)	1302h	AH = 13h AL = 02h BH = ページ番号(0を設定) CX = 書き込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP = 書き込み文字・属性列 ※ 文字列は文字コード属性を連続で指定		P.42
	文字・属性列の書き込み (カーソル移動あり)	1303h	AH = 13h AL = 03h BH = ページ番号(0を設定) CX = 書き込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP = 書き込み文字・属性列 ※文字列は文字コード·属性を連続で指定		P.42
	文字・属性列の取得 (カーソル移動なし)	1310h	AH = 13h AL = 10h BH = ページ番号(0を設定) CX = 読み込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP=読み込み文字·属性列格納アドレス ※文字列は文字コード·属性を連続で格納		P.40
	文字・拡張属性列の取得 (カーソル移動なし)	1311h	AH = 13h AL = 11h BH = ページ番号(0を設定) CX = 読み込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP=読み込み文字·属性列格納アドレス ※ 文字列は文字コード·属性0・属性1・属性2を連続で格納		P.40
	文字・属性列の書き込み (カーソル移動なし)	1320h	AH = 13h AL = 20h BH = ページ番号(0を設定) CX = 書き込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP = 書き込み文字・属性列 ※文字列は文字コード·属性を連続で指定		P.42
	文字・拡張属性列の書込み (カーソル移動なし)	1321h	AH = 13h AL = 21h BH = ページ番号(0を設定) CX = 書き込み文字数 DH = 0からの相対行位置 DL = 0からの相対桁位置 ES:BP = 書き込み文字・属性列 ※ 文字列は文字コード·属性0・属性1・属性2を連続で指定		P.42
	ユーザーフォントパターンの取得	1800h	AH = 18h AL = 00h	AL = 00h ····· 正常終了 != 00h ····· エラー	P.57

割り込み	機　能　名	機能番号	入　　力	出　　力	頁番号
INT 10h			BH = 0 BL = 0 CH = 全角上位1バイト目 半角なら00h CL = 全角下位2バイト目 半角文字コード DH = 登録フォントのドット幅数 DL = 登録フォントのライン行数 ES:SI = 登録フォントのイメージ格納バッファ	ES:SI=登録フォントのイメージ格納バッファ	
	ユーザーフォントパターンの登録	1801h	AH = 18h AL = 01h BH = 0 BL = 0 CH = 全角上位1バイト目 CL = 全角下位2バイト目 DH = 登録フォントのドット幅数 DL = 登録フォントのライン行数 ES:SI = 登録フォントのイメージ格納バッファ	AL = 00h ····· 正常終了 != 00h ····· エラー	P.57
	ディスプレイ組合せコード取得	1A00h	AH = 1Ah	AL = 1Ah ······ 機能が対応 != 1Ah ······ 機能未サポート BL = 現在アクティブな画面コード 00h　:ディスプレイなし 01h～06h:予約済み 07h　:VGA(ｱﾅﾛｸﾞ単色) 08h　:CGA(カラー)	
	シフトキー状態表示域の表示	1D00h	AH = 1Dh AL = 00h BX = 状態表示域の行数		
	シフトキー状態表示域の消去	1D01h	AH = 1Dh AL = 01h		
	シフトキー状態表示域の状態取得	1D02h	AH = 1Dh AL = 02h	BX = 状態表示域の行数	P.32
	VRAMアドレスの取得	FEh	AH = FEh ES = B800h DI = 0000h	ES = VRAM セグメントアドレス DI = VRAM オフセットアドレス	P.44
	表示画面の更新	FFh	AH = FFh CX = 更新する文字数 ES:DI = 更新開始アドレス		P.44
INT 11h	装置構成情報の取得			F E D C B A 9 8 7 6 5 4 3 2 1 0 AX [ ][ ][X][X][-][-][-][X][-][-][X][X][X][1][1][1] F-E: 接続プリンタ数 B-9: ASYNC 通信ポート数 7-6: 00: 1ドライブ　01: 2ドライブ 2: マウス接続 1: コプロ装備	
INT 12h	基本メモリ容量の取得			AX = 1KB単位の連続した空き容量	
(FDD) INT 13h	ディスクシステムのリセット	00h	AH = 00h DL = ドライブ番号(0=A:,1=B:,2=C:....)	AH = ディスクステータス 00h: エラーなし 01h: 無効なディスケットパラメータ 02h: アドレスマークが無い 03h: 書き込み禁止 04h: セクタが見つからない 05h: リセット失敗 06h: ディスケットが交換された 07h: パラメータテーブルが不良 08h: DMAオーバーラン発生 09h: DMAで64Kを越えた 0Ah: セクタの不良 0Bh: シリンダが不良 0Ch: メディアタイプが不正 0Dh: 初期化時のセクタ数不良 0Eh: 制御データアドレスマーク検出 0Fh: DMA処理レベルが越えた 10h: リードCRCまたはECCエラー 11h: ECCによってデータエラー補正 20h: コントローラ異常 40h: シークエラー 80h: タイムアウトエラー AAh: ドライブの準備ができてない BBh: 未定義エラー CCh: 書き込みエラー E0h: ステータスエラー	

割り込み	機能名	機能番号	入力	出力	頁番号
(FDD) INT 13h				FFh: 検出処理エラー	
	ディスクステータスの取得	01h	AH = 01h DL = ドライブ番号(0=A:,1=B:,2=C:....)	AH = ディスクステータス (機能00hを参照)	
	セクタの読み込み	02h	AH = 02h AL = 読み込むセクタ数 CH = シリンダ番号 CL = セクタ番号 DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....) ES:BX = 読み込みバッファアドレス	AH = ディスクステータス (機能00hを参照) AL = 読み込んだセクタ数	
	セクタの書き込み	03h	AH = 03h AL = 書き込むセクタ数 CH = シリンダ番号 CL = セクタ番号 DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....) ES:BX = 書き込みバッファアドレス	AH = ディスクステータス (機能00hを参照) AL = 書き込んだセクタ数	
	セクタのベリファイ	04h	AH = 04h AL = 確認するセクタ数 CH = シリンダ番号 CL = セクタ番号 DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....) ES:BX = 確認データバッファアドレス	AH = ディスクステータス (機能00hを参照) AL = 確認したセクタ数	
	トラックのフォーマット	05h	AH = 05h ES:BX = 4バイトアドレスフィールドのアドレス	AH = ディスクステータス (機能00hを参照)	
	ドライブパラメータの取得	08h	AH = 08h DL = ドライブ番号(0=A:,1=B:,2=C:....)	AX = 0 BH = 0 BL = ドライブタイプ CH = 最大シリンダ数 CL = 最大セクタ/トラック DH = 最大ヘッド数 DL = ドライブ数 ES:DI = 11バイトパラメータテーブルのアドレス	
	ディスクドライブタイプの取得	15h	AH = 15h DL = ドライブ番号(0=A:,1=B:,2=C:....)	AH = タイプ 00h: 存在しないドライブ 01h: チェンジライン非サポート 02h: チェンジライン 03h: 固定ディスク	
	ディスケットの差し替え状態取得	16h	AH = 16h DL = ドライブ番号(0=A:,1=B:,2=C:....)	AH = 差し替え状態 00h: 差し替え信号の検知なし 01h: 無効なディスクパラメータ 06h: 差し替え信号を検知 80h: ディスケットドライブ動作不能	
	メディアタイプの設定	18h	AH = 18h CH = トラック数 CL = セクタ/トラック情報 7 6 5 4 3 2 1 0 ビット7-6: トラック数10ビット中の上位2ビット ビット5-0: トラック当りのセクタ数 DL = ドライブ番号(0=A:,1=B:,2=C:....)	AH = ディスケットステータス ES:DI = 11バイトパラメータテーブルのアドレス	
	メディアタイプの取得	20h	AH = 20h DL = ドライブ番号(0=A:,1=B:,2=C:....)	AH = ディスケットの状態 01h: 無効なコマンド 30H: メディアセンスが未 31H: メディアが未挿入 32H: メディアタイプが不正 AL = メディアタイプ 06h: 4MBディスケット 04h: 2MBディスケット 03h: 1MBディスケット	
(HDD) INT 13h	ディスクシステムのリセット	00h	AH = 00h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AH = ディスクステータス 00h: エラーなし 01h: 無効なディスケットパラメータ 02h: アドレスマークが無い 03h: 書き込み禁止 04h: セクタが見つからない 05h: リセット失敗 06h: ディスケットが交換された 07h: パラメータテーブルが不良 08h: DMAオーバーラン発生	

割り込み	機能名	機能番号	入力	出力	頁番号
(HDD) INT 13h				09h: DMAで64Kを越えた 0Ah: セクタの不良 0Bh: シリンダが不良 0Ch: メディアタイプが不正 0Dh: 初期化時のセクタ数不良 0Eh: 制御データアドレスマーク検出 0Fh: DMA処理レベルが越えた 10h: リードCRCまたはECCエラー 11h: ECCによってデータエラー補正 20h: コントローラ異常 40h: シークエラー 80h: タイムアウトエラー AAh: ドライブの準備ができてない BBh: 未定義エラー CCh: 書き込みエラー E0h: ステータスエラー FFh: 検出処理エラー	
	ディスクステータスの取得	01h	AH = 01h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB(80h)をONにして指定	AH = ディスクステータス (機能00hを参照)	
	セクタの読み込み	02h	AH = 02h AL = 読み込むセクタ数 CH = シリンダ番号 CL = セクタ番号 7 6 5 4 3 2 1 0 (ビット7-6: シリンダ番号10ビット中の上位2ビット、ビット5-0: トラック番号) DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定 ES:BX = 読み込みバッファアドレス	AH = ディスクステータス (機能00hを参照) AL = 読み込んだセクタ数	
	セクタの書き込み	03h	AH = 03h AL = 書き込むセクタ数 CH = シリンダ番号 CL = セクタ番号 「セクタの読み込み(機能02h)」を参照 DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定 ES:BX = 書き込みバッファアドレス	AH = ディスクステータス (機能00hを参照) AL = 書き込んだセクタ数	
	セクタのベリファイ	04h	AH = 04h AL = 確認するセクタ数 CH = シリンダ番号 CL = セクタ番号 「セクタの読み込み(機能02h)」を参照 DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定 ES:BX = 確認データバッファアドレス	AH = ディスクステータス (機能00hを参照) AL = 確認したセクタ数	
	シリンダの初期化	05h	AH = 05h AL = セクタ数 CX = シリンダ番号 DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定 ES:BX = 512バイトアドレスフィールドのアドレス	AH = ディスクステータス (機能00hを参照)	
	ドライブパラメータの取得	08h	AH = 08h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AX = 0 BH = 0 BL = ドライブタイプ CH = シリンダ数 CL = セクタ 「セクタの読み込み(機能02h)」参照 DH = 最大ヘッド数 DL = ドライブ数 ES:DI =11バイトパラメータテーブルのアドレス	
	ドライブ初期化	09h	AH = 09h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AH = ディスクステータス (機能00hを参照)	
	シーク処理	0Ch	AH = 0Ch CX = シリンダ番号 DH = ヘッド番号 DL = ドライブ番号(0=A:,1=B:,2=C:....)	AH = ディスクステータス (機能00hを参照)	

割り込み	機 能 名	機能番号	入 力	出 力	頁番号
(HDD) INT 13h			MSB(80h)をONにして指定		
	代替ディスクリセット	0Dh	AH = 0Dh DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AH = ディスクステータス (機能00hを参照)	
	ドライブ状態の検査	10h	AH = 10h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AH = ディスクステータス (機能00hを参照)	
	ヘッドの再設定	11h	AH = 11h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AH = ディスクステータス (機能00hを参照)	
	ハードディスクタイプの取得	15h	AH = 15h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AH = タイプ 00h: 存在しないドライブ 01h: チェンジライン非サポート 02h: チェンジラインサポート 03h: 固定ディスク CX =512バイトブロック総数の上位 DX =512バイトブロック総数の下位	
	ヘッドパーク	19h	AH = 19h DL = ドライブ番号(0=A:,1=B:,2=C:....) MSB (80h) をONにして指定	AH = ディスクステータス (機能00hを参照)	
INT 14h	通信ポートの初期化	00h	AH = 00h AL = ポート設定情報 7 6 5 4 3 2 1 0 データビット長 ストップビット パリティ ボーレート DX = ポート番号	AH = 回線状態 7 6 5 4 3 2 1 0 データレディ オーバーランエラー パリティエラー フレームエラー ブレーク信号検出 送信用保持レジスタ空きなし 送信用シフトレジスタ空きなし タイムアウト AL = モデム状態 7 6 5 4 3 2 1 0 送信可に変化 DSR に変化 呼び出し信号の終 受信回線信号の検出に変化 送信可(CTS) DSR 呼び出し信号受信 受信回線信号の検出	
	1文字送信	01h	AH = 01h AL = 送信文字 DX = ポート番号	AH = 回線状態 (機能00hを参照)	
	1文字受信	02h	AH = 02h DX = ポート番号	AL = 受信した文字 AH = 回線状態 (機能00hを参照)	
	通信ポート状態の取得	03h	AH = 03h DX = ポート番号	AH = 回線状態 AL = モデム状態 (機能00hを参照)	
	通信ポートの初期化(拡張)	04h	AH = 04h AL = ブレーク設定 00h: ブレークなし 01h: ブレークあり BH = パリティ 00h: なし 01h: 奇数 02h: 偶数 03h: スティックパリティ奇数 04h: スティックパリティ偶数 BL = ストップビット 00h: 1ビット 01h: 2または1.5ビット CH = ワード長 00h: 5ビット 01h: 6ビット 02h: 7ビット 03h: 8ビット CL = ボーレート 00h: 110 bps 01h: 150 bps 02h: 300 bps 03h: 600 bps 04h: 1200 bps 05h: 2400 bps	AH = 回線状態 AL = モデム状態 (機能00hを参照)	

割り込み	機能名	機能番号	入力	出力	頁番号
INT 14h			06h: 4800 bps 07h: 9600 bps 08h: 19200 bps		
	モデム制御レジスタの取得	0500h	AH = 05h AL = 00h DX = ポート番号	BL = モデム制御レジスタ 7 6 5 4 3 2 1 0 0: データレディ 1: 送信要求(RST) 2: OUT 1 3: OUT 2 4: ループ 5: 予約 6: 予約 7: 予約	
	モデム制御レジスタの書き込み	0501h	AH = 05h AL = 01h BL = モデム制御レジスタ DX = ポート番号	AH = 回線状態 AL = モデム状態 (機能00hを参照)	
INT 15h	BIOSタイプの取得	49h	AH = 49H AL = 00h	BL = BIOS タイプ 00h: DOS/V BIOS or PS/2 BIOS 01h: 日本語DOS K3.4以下 or DOS J4.0以上のBIOS AH = 状況 CF=1:AH = 86h 当機能は未サポート CF=0:AH = 00h 当機能はサポートされている	P.16
	キーボードインターセプト	4Fh	AH = 4Fh AL = 走査コード キーが押されたときにシステムから呼び出される	CF=1: AL = 新しい走査コード CF=0: AL = 走査コードは変更なし	P.100
	フォントの読み取り処理ルーチンアドレスの取得	5000h	AH = 50h AL = 00h BH = フォントの種類 7 6 5 4 3 2 1 0 ビット0: 0:1バイト文字セット 1:2バイト文字セット ビット1〜7: 予約 BL = 00h DH = フォント幅 DL = フォント高 BP = コードページ	CF = 処理結果 0 : 正常 1 : 異常 AH = 完了コード 00h: 正常(CF=0の場合) 01h: 無効なフォント種類 02h: BL が0ではない 03h: 無効なフォントサイズ 04h: 無効なコードページ 86h: 当機能は未サポート ES:BX = 処理ルーチンアドレス	
	フォントの書き込み処理ルーチンアドレスの取得	5001h	AH = 50h AL = 01h BH = フォントの種類 7 6 5 4 3 2 1 0 ビット0: 0:1バイト文字セット 1:2バイト文字セット ビット1〜7: 予約 BL = 00h DH = フォント幅 DL = フォント高 BP = コードページ	CF = 処理結果 0 : 正常 1 : 異常 AH = 完了コード (機能5000hを参照) ES:BX = 処理ルーチンアドレス	
	フォントの読み込みと登録	ES:BX	CX = 文字コード(半角時,CH=00h) ES:SI = 文字バッファ	AL = 完了コード 00h: 正常 05h: 無効な文字コード 06h: 指定されたフォントは読込専用	
	イベントウェイトの設定	8300h	AH = 83h AL = 00h ES:BX = 完了フラグのアドレス CX:DX = μs単位の待ち時間	CF = 処理結果 0 : 正常 1 : すでに設定済み	
	イベントウェイトの取消	8301h	AH = 83h AL = 01h	CF = 処理結果 0 : 正常 1 : 異常	
	SysRqキーによる呼び出し	85h	AH = 85h AL = 00h MAKE = 01h BREAK SysRqキーが押されたときにシステムから呼び出される		
	待ち時間の経過待ち	86h	AH = 86h CX:DX = μs単位の待ち時間	CF = 処理結果 0 : 正常	

割り込み	機　能　名	機能番号	入　　力	出　　力	頁番号
INT 15h				1：すでにウェイト中	
	保護モード用ブロック転送	87h	AH = 87h CX = 転送ブロックサイズ(MAX=8000h) ES:SI = GDT アドレス	AH = 完了コード 00h: 正常 01h: RAMパリティエラー 02h: 他の例外割り込み発生 03h: ゲートアドレスライン20h失敗	
	100000h 以上の拡張メモリ サイズの判別	88h	AH = 88h	AX = 連続する1K単位のブロック数	
	ポインティングデバイス状態設定 [PS/2]	C200h	AH = C2h AL = 00h BH = 設定情報 00h: 使用不可 01h: 使用可能	CF = 0 正常に終了 AH = 完了コード 00h: 正常 01h: 無効な機能呼び出し 02h: 無効な入力 03h: インターフェースエラー 04h: 再送 05h: far call が未設定	
	ポインティングデバイスのリセット [PS/2]	C201h	AH = C2h AL = 01h	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照) BL = 装置ID	
	抽出率の設定 [PS/2]	C202h	AH = C2h AL = 02h BH = 抽出率 00h: 1秒当たり10回 01h: 1秒当たり20回 02h: 1秒当たり40回 03h: 1秒当たり60回 04h: 1秒当たり80回 05h: 1秒当たり100回 06h: 1秒当たり200回	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照) BL = 装置ID	
	解像度の設定 [PS/2]	C203h	AH = C2h AL = 03h BH = 解像度 00h: 1mm当たり1カウント 01h: 1mm当たり2カウント 02h: 1mm当たり4カウント 03h: 1mm当たり8カウント	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照) BL = 装置ID	
	装置タイプの取得 [PS/2]	C204h	AH = C2h AL = 04h	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照) BL = 装置ID	
	ポインティングデバイスインターフェースの初期化 [PS/2]	C205h	AH = C2h AL = 05h BH = データのパックサイズ 00h: 予約 01h: 1バイト 02h: 2バイト 03h: 3バイト 04h: 4バイト 05h: 5バイト 06h: 6バイト 07h: 7バイト 08h: 8バイト	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照) BL = 装置ID	
	ポインティングデバイスインターフェース状態の取得 [PS/2]	C206h 00h	AH = C2h AL = 06h BH = 00h	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照) BL = 状態バイト 7 6 5 4 3 2 1 0 ビット0: 右ボタン押下 ビット1: 予約 ビット2: 左ボタン押下 ビット3: 予約 ビット4: 0:縮尺率1:1 / 1:縮尺率2:1 ビット5: 使用可否 ビット6: 0: ストリームモード / 1: リモートモード CL = 状態バイト2 00h: 1mm当たり1カウント 01h: 1mm当たり2カウント 02h: 1mm当たり4カウント 03h: 1mm当たり8カウント DL = 状態バイト3 0Ah: 1秒当たり10回 14h: 1秒当たり20回 28H: 1秒当たり40回	

割り込み	機能名	機能番号	入力	出力	頁番号
INT 15h				3CH: 1秒当たり60回 50h: 1秒当たり80回 64H: 1秒当たり100回 C8H: 1秒当たり200回	
	ポインティングデバイス インターフェース 縮尺率を1:1に設定 [PS/2]	C206h 01h	AH = C2h AL = 06h BH = 01h	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照)	
	ポインティングデバイス インターフェース 縮尺率を2:1に設定 [PS/2]	C206h 02h	AH = C2h AL = 06h BH = 02h	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照)	
	デバイスドライバfar call 初期設定 [PS/2]	C207h	AH = C2h AL = 07h ES:BX = far call ルーチンアドレス	CF = 0 正常に終了 AH = 完了コード (機能C200hを参照)	
INT 16h	1文字取得	00h	AH = 00h	AH = 走査コード AL = 文字コード	P.95
	文字入力センス	01h	AH = 01h	ZF = センス状態 0: 入力文字あり 1: 入力文字なし	P.95
	ソフト押下状態の取得	02h	AH = 02h	AL = 現在のシフト押下状態 7 6 5 4 3 2 1 0 右シフト押下 左シフト押下 Ctrlキー押下 Altキー押下 ScrollLock NumLock CapsLock 挿入モード	P.87
	キータイプマチック速度 の設定	03h	AH = 03h AL = 05h BL = タイプ可能文字数/1秒当たり 00h: 30.0 01h: 26.7 02h: 24.0 03h: 21.8 04h: 20.0 05h: 18.5 06h: 17.1 07h: 16.0 08h: 15.0 09h: 13.3 0Ah: 12.0 0Bh: 10.9 0Ch: 10.0 0Dh: 9.2 0Eh: 8.6 0Fh: 8.0 10h: 7.5 11h: 6.7 12h: 6.0 13h: 5.5 14h: 5.0 15h: 4.6 16h: 4.3 17h: 4.0 18h: 3.7 19h: 3.3 1Ah: 3.0 1BH: 2.7 1CH: 2.5 1Dh: 2.3 1EH: 2.1 1FH: 2.0 BH = 遅延時間 00h: 250 ms 01h: 500 ms 02h: 750 ms 03h: 1000 ms		P.97
	キーバッファへの書き込み	05h	AH = 05h CL = 文字コード CH = 走査コード	AL = 完了コード 00h: 正常 01h: バッファフル	P.99
	キーボードIDの取得	0Ah	AH = 0Ah	BX = キーボードID 0000h: キーボード未接続 84ABH: 5535-S 型 54ABH: 5535-S 型 83ABH: 5576-A01, 5535-S 型	

割り込み	機　能　名	機能番号	入　　力	出　　力	頁番号
				（テンキーパッド付） 41ABH: 5576-A01,5535-S 型 （テンキーパッド付） 90ABH: 5576-002 型 91ABH: 5576-003 型 92ABH: 5576-001 型	
	拡張1文字取得	10h	AH = 10h	AH = 走査コード AL = 文字コード	P.95
	拡張文字入力センス	11h	AH = 11h	ZF = センス状態 0: 入力文字あり 1: 入力文字なし	P.95
	拡張ソフト押下状態の取得	12h	AH = 12h	AL = 現在のシフト押下状態 7 6 5 4 3 2 1 0 0: 右シフト押下 1: 左シフト押下 2: Ctrlキー押下 3: Altキー押下 4: ScrollLock 5: NumLock 6: CapsLock 7: 挿入モード AH = 拡張シフト押下状態 7 6 5 4 3 2 1 0 0: 左Ctrlキー押下 1: 左Altキー押下 2: 右Ctrlキー押下 3: 左Altキー押下 4: ScrollLock 5: NumLock 6: CapsLock 7: SysReq	P.87
	DBCS 状態の設定	1300h	AH = 13h AL = 00h DH = (1300h で読み込んだ値) DL = 設定状態 7 6 5 4 3 2 1 0 0: 0: 半角 1:全角 1-2: 00: 英数シフト 01: カタカナシフト 10: ひらがなシフト 11: 予約 3-5: 予約 6: 0: ローマ字ON 1: ローマ字OFF 7: 0: 非漢字モード 1: 漢字モード		P.90
	DBCS 状態の取得	1301h	AH = 13h AL = 01h	DH = 予約(設定用に保持) DL = 設定状態 (機能1300hを参照)	P.90
	シフト状態の表示	1400h	AH = 14h AL = 00h		
	シフト状態の消去	1401h	AH = 14h AL = 01h		
	シフト状態の表示状況取得	1402h	AH = 14h AL = 02h	AL = 表示状況 00h: 表示中 01h: 消去中	
INT 17h	1文字の印刷	00h	AH = 00h AL = 印刷する文字 DX = 論理プリンタポート	AH = プリンタ状態 7 6 5 4 3 2 1 0 0: タイムアウト 1-2: 予約 3: I/Oエラー 4: オンライン(印刷可能) 5: 用紙切れ 6: 要求応答 7: 動作可能	

割り込み	機能名	機能番号	入力	出力	頁番号
INT 17h				[NATIVE モードの場合] 7 6 5 4 3 2 1 0 bit 0: タイムアウト bit 1–2: 予約 bit 3–5: 000: 取消しSW ON 001: 印刷不可 010: 印刷可能 011: 予約 100: 予約 101: 用紙切れ or 用紙ジャム 110: 電源OFF 111:印字ヘッド加熱 bit 6: 要求応答 bit 7: 動作可能	
	プリンタポートの初期化	01h	AH = 01h DX = 論理プリンタポート	AH = プリンタ状態 (機能番号 00h を参照)	
	プリンタ状態の取得	02h	AH = 02h DX = 論理プリンタポート	AH = プリンタ状態 (機能番号 00h を参照)	
INT 1Ah	時刻カウントの取得	00h	AH = 00h	CX = カウントの高位16ビット DX = カウントの低位16ビット AL = 24時間経過状態 0: 最後の取得から24時間未経過 !0: 24時間経過	P.161
	時刻カウントの設定	01h	AH = 01h CX = カウントの高位16ビット DX = カウントの低位16ビット		
	リアルタイムクロックの時刻取得	02h	AH = 02h	CH = 時(2進化10進数(BCD)) CL = 分(2進化10進数(BCD)) DH = 秒(2進化10進数(BCD)) DL = 00h: 夏時間制選択 01h: 夏時間制なし CF = クロック状態 0: クロック動作中 1: クロック動作してない	P.162
	リアルタイムクロックの時刻設定	03h	AH = 03h CH = 時(2進化10進数(BCD)) CL = 分(2進化10進数(BCD)) DH = 秒(2進化10進数(BCD)) DL = 00h: 夏時間制選択 01h: 夏時間制なし		
	リアルタイムクロックの日付取得	04h	AH = 04h	CH = 世紀(2進化10進数(BCD)19,20) CL = 年(2進化10進数(BCD)) DH = 月(2進化10進数(BCD)) DL = 日(2進化10進数(BCD)) CF = クロック状態 0: クロック動作中 1: クロック動作してない	P.162
	リアルタイムクロックの日付設定	05h	AH = 05h CH = 世紀(2進化10進数(BCD)19,20 CL = 年(2進化10進数(BCD)) DH = 月(2進化10進数(BCD)) DL = 日(2進化10進数(BCD))		
	アラームの設定	06h	AH = 06h CH = 時(2進化10進数(BCD)) CL = 分(2進化10進数(BCD)) DH = 秒(2進化10進数(BCD))	CF = 処理結果 0: 正常終了 1: 異常終了	P.163
	アラームの取消	07h	AH = 07h	なし	P.163
INT 1Bh	Ctrl+Break割り込み				
INT 1Ch	タイマ割り込み				

資料編2

BIOSワークエリア

セグメント	:	オフセット	サイズ	内　容
40h	:	00h	WORD	シリアルポート#1・ベースアドレス(0のときは未接続)
40h	:	02h	WORD	シリアルポート#2・ベースアドレス(0のときは未接続)
40h	:	04h	WORD	シリアルポート#3・ベースアドレス(0のときは未接続)
40h	:	06h	WORD	シリアルポート#4・ベースアドレス(0のときは未接続)
40h	:	08h	WORD	パラレルポート#1・ベースアドレス(0のときは未接続)
40h	:	0Ah	WORD	パラレルポート#2・ベースアドレス(0のときは未接続)
40h	:	0Ch	WORD	パラレルポート#3・ベースアドレス(0のときは未接続)
40h	:	0Eh	WORD	[AT] パラレルポート#4・ベースアドレス(0のときは未接続) [PS/2] 拡張BIOSデータセグメントのセグメント
40h	:	10h	WORD	装置構成情報(INT 11hと同じ) ビット15-14　プリンタポート数 ビット13　予約済み ビット12　(ジョイスティック数) ビット11-9　シリアルポート数 ビット8　予約済み ビット7-6　接続ディスケットドライブ数 - 1 ビット5-4　ビデオモード 00 = EGA,VGA,PGA 01 = 40×25 カラー 10 = 80×25 カラー 11 = 80×25 モノクロ ビット3　予約済み ビット2　[PS/2] = 1 マウスが接続 [AT] 予約済み ビット1　= 1 数値演算コプロセッサ実装 ビット0　= 1 ディスケットからブート可能
40h	:	12h	BYTE	予約済み
40h	:	13h	WORD	基本メモリサイズ(0～640KB)
40h	:	15h	BYTE	予約済み
40h	:	16h	BYTE	予約済み
40h	:	17h	BYTE	キーボードシフトステータス(PC86キー) ビット7 = 1 挿入状態 ビット6 = 1 CapsLock状態 ビット5 = 1 NumLock状態 ビット4 = 1 ScrollLock状態 ビット3 = 1 Alt(前面)キー押下 ビット2 = 1 Ctrlキー押下 ビット1 = 1 左Shiftキー押下 ビット0 = 1 右Shiftキー押下
40h	:	18h	BYTE	キーボードシフトステータス(AT101キー) ビット7 = 1 Insertキー押下 ビット6 = 1 CapsLockキー押下 ビット5 = 1 NumLockキー押下 ビット4 = 1 ScrollLockキー押下 ビット3 = 1 Pause状態 ビット2 = 1 SysRqキー押下 ビット1 = 1 左Alt(前面)キー押下 ビット0 = 1 左Ctrlキー押下
40h	:	19h	BYTE	キーボード: テンキーパッド作業空間
40h	:	1Ah	WORD	キーボード: キーバッファ内の読み取り位置ポインタ
40h	:	1Ch	WORD	キーボード: キーバッファ内の書き出し位置ポインタ
40h	:	1Eh	16WORD	キーボード: キーボードリングバッファ(80h,82hで置き替え可能)
40h	:	3Eh	BYTE	ディスケットドライブ：リキャブレートフラグ (ドライブヘッドを最外周へ移動する) ビット7 = 1 ディスケットハードウェア割り込みが発生した ビット6-4　予約済み ビット3 = 1 ドライブ3：リキャブレート ビット2 = 1 ドライブ2：リキャブレート

セグメント：オフセット	サイズ	内容
		ビット1 = 1 ドライブ1：リキャブレート
		ビット0 = 1 ドライブ0：リキャブレート
40h : 3Fh	BYTE	ディスケットドライブ：モータの状態
		ビット7 = 1 現在の動作状態：書き込みまたはフォーマット中
		= 0 現在の動作状態：読み込みまたはベリファイ中
		ビット6 予約済み
		ビット5-4 選択されているドライブ(0～3)
		ビット3 = 1 ドライブ3モータ作動
		ビット2 = 1 ドライブ2モータ作動
		ビット1 = 1 ドライブ1モータ作動
		ビット0 = 1 ドライブ0モータ作動
40h : 40h	BYTE	ディスケットドライブ：モータ停止用カウンタ
		(55ms単位にカウンタを減算し，0になるとモータを停止する)
40h : 41h	BYTE	直前のディスク処理結果状態
		00h 正常終了
		01h 無効なパラメータが渡された
		02h アドレスマークが見つからない
		03h 書き込み禁止ディスケットへの書き込み
		04h 要求されたセクタが見つからない
		06h ディスケットが交換された
		08h 操作中にDMAのオーバーランが発生した
		09h 64KB境界をまたがるDMAアクセス
		0Ch 不正なメディアタイプ
		10h 読み込みでCRCエラー
		20h ディスケットコントローラの障害
		40h シーク操作の失敗
		80h ドライブの準備ができていない
40h : 42h	7BYTE	ディスケットコントローラ：ステータス
40h : 49h	BYTE	現在のビデオモード
40h : 4Ah	WORD	現在の画面表示桁数
40h : 4Ch	WORD	1画面の文字数(バイト数)
40h : 4Eh	WORD	ビデオRAM開始オフセットアドレス
40h : 50h	16BYTE	0～7ページまでの，カーソル位置
		1バイト目：現在の桁位置
		2バイト目：現在の行位置
40h : 60h	WORD	カーソルの形状
		1バイト目：カーソル終了位置
		2バイト目：カーソル開始位置
40h : 62h	BYTE	活動ページ番号(現在表示されている画面ページ番号)
		DOS/V日本語モードでは1ページしかサポートしていないため
		常に0が設定されています．
40h : 63h	WORD	CRTコントローラのI/Oアドレス
		カラー：03D4h モノクロ：03B4h
40h : 65h	BYTE	CRTモード選択レジスタ(03D8h/03B8h)の現在値
40h : 66h	BYTE	CGAパレットレジスタ(03D9h)の現在値
40h : 67h	DWORD	システムデータエリア1：予約済み
40h : 6Bh	BYTE	システムデータエリア1：予約済み
40h : 6Ch	DWORD	深夜0時を起点とする加算カウンタ．24時で0にリセットされる．
		(システムタイマ)
40h : 70h	BYTE	システムタイマが0にリセットされると，1になり
		日付更新が必要である事を知らせるフラグ．
		(システムタイマオーバーフロー)
40h : 71h	BYTE	Ctrl+Breakキーが押されると，ビット7が1になる．
40h : 72h	WORD	リセットフラグ
		1234h ： ウォームブート(メモリテスト省略)
		4321h ： PS/2または，MCAのみで使用．メモリ内容を保持．
40h : 74h	BYTE	直前のディスク操作状況(EDSIの除く)
		00h 正常終了
		01h 無効なパラメータが渡された
		02h アドレスマークが見つからない
		04h 要求されたセクタが見つからない
		05h リセットに失敗した
		07h ドライブパラメータの異常
		08h 操作中にDMAのオーバーランが発生した
		09h 64KB境界をまたがるDMAアクセス
		0Ah 不良セクタフラグ検出
		0Bh 不良シリンダ検出
		0Dh フォーマットにおける無効なセクタ数
		0Eh 制御データアドレスマーク検出
		0Fh DMAのアビトレーションレベルが範囲外
		10h ECCの訂正不可，またはCRCエラーが発生
		11h ECCの訂正データエラー
		12h コマンド処理中
		13h ドライブ電源断

セグ　　オフ
メント：セット サイズ 内　容

20h ディスクコントローラの障害
40h シーク操作の失敗
80h タイムアウト発生
AAh ドライブの準備ができていない
BBh 未定義エラーが発生
CCh 選択ドライブでの書き込み失敗
E0h 状況エラー/エラーレジスタが0
FFh センス操作の失敗

40h : 75h BYTE 接続ハードディスク数
40h : 76h BYTE ハードディスク制御バイト(XT専用)
40h : 77h BYTE ハードディスクI/Oポートアドレス(XT専用)
40h : 78h 4 BYTE パラレルポートタイムアウト時間×4
PS/2システムの場合，パラレルは0-2までしか使用していないので，先頭の3バイトのみ使用となる．
40h : 7Ch 4 BYTE シリアルポートタイムアウト時間×4
40h : 80h WORD キーバッファの先頭オフセット(通常は1Eh)
40h : 82h WORD キーバッファの末尾オフセット(通常は3Eh)
40h : 84h BYTE EGA/MCGA/VGA 表示可能行数-1
40h : 85h WORD EGA/MCGA/VGA キャラクタの高さを走査線数でセット
40h : 87h BYTE EGA/VGA 制御: MCGA=00h

ビット7 = 1 VRAMをクリアしない(INT 10h, AH=00h参照)
ビット6-5 ビデオRAM容量
00 : 64KB
01 : 128KB
10 : 192KB
11 : 256KB
ビット4 予約済み
ビット3 = 0 EGA/VGAが動作
= 1 EGA/VGAが動作していない
ビット2 = 1 予約済み
ビット1 = 0 カラーまたはECDモニター
= 1 モノクロモニター
ビット0 = 0 英数字カーソルエミュレーション不可
= 1 英数字カーソルエミュレーション可

40h : 88h BYTE EGA/VGA スイッチ状態: MCGA:予約

ビット7-4 電源投入時のEGA フィーチャコネクター状態
ビット3-0 コンフィグレーションスイッチ4-1(=0 ON, =1 OFF)
この下位4ビットは以下のとおり見ることができます．
0h MDA，もしくはEGA+旧カラー画面40×25
1h MDA，もしくはEGA+旧カラー画面80×25
2h MDA，もしくはEGA+ECD通常モード
(CGAエミュレーション)
3h MDA，もしくはEGA+ECD拡張モード
4h CGA 40×25，もしくはEGAモノクロ画面
5h CGA 80×25，もしくはEGAモノクロ画面
6h EGA+旧カラー画面40×25，もしくはMDA
7h EGA+旧カラー画面80×25，もしくはMDA
8h EGA+ECD通常モード(CGAエミュレーション)，
もしくはMDA
9h EGA+ECD拡張モード，もしくはMDA
Ah EGAモノクロ画面，もしくはCGA 40×25
Bh EGAモノクロ画面，もしくはCGA 80×25

40h:89h のビット4が0のとき，当バイトがx3hまたはx9hの場合は，VGAは350ラインEGAをエミュレーションしている．
そうでなければ，400ラインダブルスキャンの200ラインCGAをエミュレーションしている．VGAは当バイトを，画面モード設定後にx9hにセットする．

40h : 89h BYTE MCGA/VGA モードセットオプション制御:

ビット7と4
0 0 350ラインモード
0 1 400ラインモード
1 0 200ラインモード
1 1 予約済み
ビット6 = 1 ディスプレースイッチイネーブル
= 0 ディスプレースイッチディスエーブル
ビット5 予約済み
ビット3 = 0 モードセット時にデフォルトパレットをロードする
ビット2 = 1 モノクロディスプレイ
= 0 カラーディスプレイ
ビット1 = 1 グレースケールサミングイネーブル
= 0 グレースケールサミングディスエーブル
ビット0 [VGA] = 1 VGAがアクティブ
= 0 アクティブでない
[MCGA] 予約済み(0)

セグメント	オフセット	サイズ	内　容
40h	8Ah	BYTE	[MCGA/VGA] ディスプレイコンビネーションコードテーブルのインデックス
40h	8Bh	BYTE	[XT以外] ディスケットメディア制御
			ビット7-6　最後に操作したドライブのデータ転送レート 00=500KB/S, 01=300KB/S, 10=250KB/S, 11=予約済み
			ビット5-4　最後に操作したドライブのステッププレート
			ビット3-0　予約済み
40h	8Ch	BYTE	[XT以外] ディスク制御状態
40h	8Dh	BYTE	[XT以外] ディスク制御エラー状態
40h	8Eh	BYTE	[XT以外] ディスク割り込み制御
40h	8Fh	BYTE	[XT以外] ディスケットコントローラ情報
			ビット7　　予約済み
			ビット6 = 1 ドライブ1が接続?
			ビット5 = 1 ドライブ1がマルチレート
			ビット4 = 1 ドライブ1は80トラックをサポートしている
			ビット3　　予約済み
			ビット2 = 1 ドライブ0が接続?
			ビット1 = 1 ドライブ0がマルチレート
			ビット0 = 1 ドライブ0は80トラックをサポートしている
40h	90h	BYTE	ディスケットドライブ0のメディア状態
40h	91h	BYTE	ディスケットドライブ1のメディア状態
			ビット7-6　データレート: 00=500KB/S, 01=300KB/S, 10=250KB/S
			ビット5 = 1 ダブルステッピングが必要(e.g. 360KB in 1.2MB)
			ビット4 = 1 メディア確立
			ビット3　　予約済み
			ビット2-0　メディア/ドライブ状況
			000 360KB/360KBトライ中
			001 360KB/1.2MBトライ中
			010 1.2MB/1.2MBトライ中
			011 360KB/360KB確立
			100 360KB/1.2MB確立
			101 1.2MB/1.2MB確立
			110 予約済み
			111 その他のメディア/ドライブ
40h	92h	BYTE	ディスケットドライブ0:操作開始時のメディア状況
40h	93h	BYTE	ディスケットドライブ1:操作開始時のメディア状況
40h	94h	BYTE	ディスケットドライブ0:現在のシリンダ番号
40h	95h	BYTE	ディスケットドライブ1:現在のシリンダ番号
40h	96h	BYTE	キーボード状況バイト1
			ビット7 = 1 READ-IDコマンド実行中
			ビット6 = 1 最後のコードは, キーボードIDの1バイト目
			ビット5 = 1 READ-IDをして拡張キーボードなら, 強制Num Lock
			ビット4 = 1 拡張キーボードが接続されている
			ビット3 = 1 右Alt キー押し下げ
			ビット2 = 1 右Ctrlキー押し下げ
			ビット1 = 1 最後のコードはE0h
			ビット0 = 1 最後のコードはE1h
40h	97h	BYTE	キーボード状況バイト2
			ビット7 = 1 キーボード転送エラー
			ビット6 = 1 キーボードLED更新中
			ビット5 = 1 RESEND(FEh)コードを受け取った
			ビット4 = 1 ACK(FAh)コードを受け取った
			ビット3　　予約済み(0)
			ビット2　　Caps Lock LEDステータス
			ビット1　　Num Lock LEDステータス
			ビット0　　Scroll Lock LEDステータス
40h	98h	DWORD	リアルタイムクロック:ユーザーウェイト完了フラグへのポインタ (INT 15, AX=8300hを参照)
40h	9Ch	DWORD	リアルタイムクロック:ユーザーウェイトカウンタ(μs単位)
40h	A0h	BYTE	リアルタイムクロック:ウェイト活動中フラグ
			ビット7 = 1 ウェイト経過中
			ビット6-1　予約済み
			ビット0 = 1 INT 15h, AH=86hが実行された
40h	A1h	7BYTE	予約済み
40h	A8h	DWORD	EGA/MCGA/VGAビデオパラメータテーブルへのポインタ
40h	ACh-CDh		予約済み
40h	CEh	WORD	count of days since last boot?
40h	D0h-EFh		予約済み
40h	F0h-FFh		予約済み(アプリケーション間交信エリア)
50h	00h	BYTE	画面印刷状況
			00h 画面印刷が行われていないか. 前回, 正常に終了した.
			01h 画面印刷を実行中.
			FFh 画面印刷を実行中にエラーが発生した.
50h	04h	BYTE	1ディスケットシステムでのA/Bドライブの切り替え状況

セグメント：	オフセット	サイズ	内容
			00h ドライブAとして動作中 01h ドライブBとして動作中

資料編3

I/Oポート

以下に，I/Oポートの使用状況をまとめておきます．ただし，これらはAT互換機として一般的なものであり，組み込んでいるハードウェアや，それらのディップスイッチのセッティングなどによって一部異なる場合があります．また，プログラミング上，よく使用するポートに関しては，ビット位置に関しても解説しています．

アドレス		機能
0000h-000Fh		DMAコントローラ(0-3)
0020h,0021h		割り込みコントローラ(8259A)マスタ
0040h-0043h		システムタイマ
0040h		カウンタ0　インターバルタイマ(約18.2Hz)
0041h		カウンタ1　DRAMリフレッシュ信号発生用
0042h		カウンタ2　スピーカ用
0043h		制御用
		ビット7-6 カウンタ選択
		00=カウンタ0
		01=カウンタ1
		10=カウンタ2
		ビット5-4 読み出し指定
		00=カウンタラッチ
		01=LSBの入出力
		10=MSBの入出力
		11=LSB,MSBの順に入出力
		ビット3-1 モード指定
		000=モード0　(単発)
		001=モード1
		x10=モード2
		x11=モード3　(方形波)
		100=モード4
		101=モード5
		ビット 0　カウンタタイプ
		0=バイナリ
		1=BCD
0060h		キー入力データ
0061h		システム制御ポートB
	OUT	ビット7-4　予約済み
		ビット 3　書き込みチャネル検査
		ビット 2　書き込みパリティ検査
		ビット 1　スピーカ動作許可
		ビット 0　タイマ2出力可能
	IN	ビット 7　パリティチェック
		ビット 6　チャネルチェック
		ビット 5　タイマ2出力
		ビット 4　リフレッシュ要求フリップフロップ
		ビット 3　書き込みチャネル検査
		ビット 2　書き込みパリティ検査
		ビット 1　スピーカ動作許可
		ビット 0　タイマ2出力可能
0064h		キーボード，補助記憶装置
0070h	I/O	リアルタイム/CMOS，NMIマスク
		ビット 7　NMI割り込みの許可
		ビット 6　予約済み
		ビット5-0　CMOS-RAMのアドレス
0071h	I/O	指定アドレスのCMOS-RAMの値
0081h-0083h,0087h		DMAページレジスタ(0-3)

アドレス		機能
0089h-008Bh,008Fh		DMAページレジスタ(4-7)
00A0h,00A1h		割り込みコントローラ(8259A)スレーブ
00C0h-00DFh		DMAコントローラ(4-7)
00F0h-00F1h,00F8h-00FFh		数値演算コプロセッサ
01F0h-01F8h		ハードディスクコントローラ
0200h-0207h		ゲームI/Oアダプタ
0278h-027Ah		パラレルポート2
0278h	I/O	パラレルポートのデータ
0279h	IN	パラレルポートのステータス
		ビット 7 0 = ビジー
		ビット 6 1 = 要求した命令に対する応答
		ビット 5 1 = 用紙切れまたは自動給紙機構中のジャム
		ビット 4 1 = オンライン状態
		ビット 3 1 = I/Oエラー
		ビット 2 0 = 割り込み発生
		ビット1-0 予約済み
027Ah	OUT	パラレルポートの制御
		ビット7-6 予約済み
		ビット 5 [AT]予約済み
		[PS/2]入出力方向フラグ 0=出力, 1=入力
		ビット 4 1 = 割り込み許可
		ビット 3 1 = プリンタセレクト
		ビット 2 1 = プリンタ初期化
		ビット 1 1 = 用紙送り
		ビット 0 1 = ストローブ
02F8h-02FFh		シリアルポート2(RS-232C)
02F8h	IN	受信データレジスタ (8250B)/ 受信FIFOバッファ (16550)
	OUT	送信保持レジスタ (8250B)/ 送信FIFOバッファ (16550)
	I/O	ボーレート分周レジスタ LSB
02F9h	I/O	割り込み許可レジスタ
		ビット7-4 つねに0
		ビット 3 1 = モデムステータス割り込みを許可
		ビット 2 1 = ラインステータス割り込みを許可
		ビット 1 1 = 送信レジスタ空き割り込みを許可
		ビット 0 1 = 受信データレディ割り込みを許可
		FIFOモードではタイムアウト割り込みも許可
	I/O	ボーレート分周レジスタ MSB
02FAh	IN	割り込み識別レジスタ
		ビット7,6 11 = FIFOバッファが使用可能
		ビット5,4 つねに0
		ビット3-0 割り込み原因識別
		0001 割り込み要求なし
		0110 ラインステータス割り込み
		0100 受信データレディ割り込み
		1100 タイムアウト割り込み (FIFOモードのみ)
		0010 送信レジスタ空き割り込み
		0000 モデムステータス割り込み
	OUT	FIFO制御レジスタ (NS16550系)
		ビット7,6 受信FIFOレジスタ割り込み用トリガレベル
		00 = 1バイト
		01 = 4バイト
		10 = 8バイト
		11 = 14バイト
		ビット5-3 つねに0
		ビット 2 1 = 送信FIFOバッファのリセット
		ビット 1 1 = 受信FIFOバッファのリセット
		ビット 0 0 = 文字モード
		1 = FIFOモード
02FBh	I/O	ライン制御レジスタ
		ビット 7 ベースアドレスおよびその次のレジスタの意味を決定
		0 = 受信データ・送信保持, 割り込み許可レジスタ
		1 = ボーレート分周レジスタ (LSB, MSBの順)
		ビット 6 ブレーク制御
		0 = 通常動作,
		1 = ブレーク送出状態
		ビット5-3 パリティチェック
		xx0 = パリティなし
		001 = 奇数パリティ
		011 = 偶数パリティ
		101 = パリティはつねに1
		111 = パリティはつねに0
		ビット 2 ストップビット
		0 = 1ストップビット

アドレス		機能
		1 = 2ストップビット
		キャラクタ長が5のときには1.5ストップビット
		ビット1,0　キャラクタ長
		00 = 5ビット
		01 = 6ビット
		10 = 7ビット
		11 = 8ビット
02FCh	I/O	モデム制御レジスタ
		ビット7-5　つねに0
		ビット 4　8250B/16550を自己診断モードにする
		ビット 3　1 = 8250B/16550の割り込み要求信号を処理する
		ビット 2　1 = Heyesコマンド内蔵モデムの電源オンリセット
		ビット 1　1 = RS(RTS)信号をオンにする
		ビット 0　1 = ER(DTR)信号をオンにする
02FDh	IN	ラインステータスレジスタ
		ビット 7　1 = FIFOバッファ中のデータにPE,FE,OEがあった
		ビット 6　1 = 送信保持レジスタ・送信シフトレジスタが空
		FIFOモード時は，送信FIFOレジスタ+シフトレジスタが空
		ビット 5　1 = 送信保持レジスタが空
		FIFOモード時は，送信FIFOレジスタが空
		ビット 4　1 = ブレーク信号を検知したとき
		ビット 3　1 = フレーミングエラー
		ビット 2　1 = パリティエラー
		ビット 1　1 = オーバーランエラー
		ビット 0　1 = 受信データレディ標識
02FEh	IN	モデムステータスレジスタ
		ビット 7　CD(DCD)のレベル
		ビット 6　CI(RI)のレベル
		ビット 5　DR(DSR)のレベル
		ビット 4　CS(CTS)のレベル
		ビット 3　CD(DCD)の変化
		ビット 2　CI(RI)の変化
		ビット 1　DR(DSR)の変化
		ビット 0　CS(CTS)の変化
02FFh	I/O	スクラッチパッドレジスタ
0378h-037Ah		パラレルポート1
0378h	I/O	パラレルポートのデータ
0379h	IN	パラレルポートのステータス
		ビット 7　0 = ビジー
		ビット 6　1 = 要求した命令に対する応答
		ビット 5　1 = 用紙切れまたは自動給紙機構中のジャム
		ビット 4　1 = オンライン状態
		ビット 3　1 = I/Oエラー
		ビット 2　0 = 割り込み発生
		ビット1-0　予約済み
037Ah	OUT	パラレルポートの制御
		ビット7-6　予約済み
		ビット 5　[AT]予約済み
		[PS/2]入出力方向フラグ　0=出力，1=入力
		ビット 4　1 = 割り込み許可
		ビット 3　1 = プリンタセレクト
		ビット 2　1 = プリンタ初期化
		ビット 1　1 = 用紙送り
		ビット 0　1 = ストローブ
03B4h,03B5h,03BAh		ビデオサブシステム
03BCh-03BEh		パラレルポート3
03BCh	I/O	パラレルポートのデータ
03BDh	IN	パラレルポートのステータス
		ビット 7　0 = ビジー
		ビット 6　1 = 要求した命令に対する応答
		ビット 5　1 = 用紙切れまたは自動給紙機構中のジャム
		ビット 4　1 = オンライン状態
		ビット 3　1 = I/Oエラー
		ビット 2　0 = 割り込み発生
		ビット1-0　予約済み
03BEh	OUT	パラレルポートの制御
		ビット7-6　予約済み
		ビット 5　[AT]予約済み
		[PS/2]入出力方向フラグ　0=出力，1=入力
		ビット 4　1 = 割り込み許可
		ビット 3　1 = プリンタセレクト

アドレス		機能
		ビット 2　1 = プリンタ初期化
		ビット 1　1 = 用紙送り
		ビット 0　1 = ストローブ
03C0h-03C5h		ビデオサブシステム
03C6h-03C9h		ビデオDAC
03CAh, 03CCh, 03CEh, 03CFh		ビデオサブシステム
03CEh		グラフィックスコントローラのアドレスレジスタ
03CFh		グラフィックスコントローラのデータレジスタ
03D4h, 03D5h, 03DAh		ビデオサブシステム
03F0h-03F7h		ディスケットドライブコントローラ
03F8h-03FFh		シリアルポート1(RS-232C)
03F8h	IN	受信データレジスタ　(8250B)/ 受信FIFOバッファ　(16550)
	OUT	送信保持レジスタ　(8250B)/ 送信FIFOバッファ　(16550)
	I/O	ボーレート分周レジスタ LSB
03F9h	I/O	割り込み許可レジスタ
		ビット7-4　つねに0
		ビット 3　1 = モデムステータス割り込みを許可
		ビット 2　1 = ラインステータス割り込みを許可
		ビット 1　1 = 送信レジスタ空き割り込みを許可
		ビット 0　1 = 受信データレディ割り込みを許可
		FIFOモードではタイムアウト割り込みも許可
	I/O	ボーレート分周レジスタ MSB
03FAh	IN	割り込み識別レジスタ
		ビット7,6　11 = FIFOバッファが使用可能
		ビット5,4　つねに0
		ビット3-0　割り込み原因識別
		0001 割り込み要求なし
		0110 ラインステータス割り込み
		0100 受信データレディ割り込み
		1100 タイムアウト割り込み (FIFOモードのみ)
		0010 送信レジスタ空き割り込み
		0000 モデムステータス割り込み
	OUT	FIFO制御レジスタ　(NS16550系)
		ビット7,6　受信FIFOレジスタ割り込み用トリガレベル
		00 = 1バイト
		01 = 4バイト
		10 = 8バイト
		11 = 14バイト
		ビット5-3　つねに0
		ビット 2　1 = 送信FIFOバッファのリセット
		ビット 1　1 = 受信FIFOバッファのリセット
		ビット 0　0 = 文字モード
		1 = FIFOモード
03FBh	I/O	ライン制御レジスタ
		ビット 7　ベースアドレスおよびその次のレジスタの意味を決定
		0 = 受信データ･送信保持,割り込み許可レジスタ
		1 = ボーレート分周レジスタ (LSB, MSBの順)
		ビット 6　ブレーク制御 0 = 通常動作,1 = ブレーク送出状態
		ビット5-3　パリティチェック
		xx0 = パリティなし
		001 = 奇数パリティ
		011 = 偶数パリティ
		101 = パリティはつねに1
		111 = パリティはつねに0
		ビット 2　ストップビット
		0 = 1ストップビット
		1 = 2ストップビット
		キャラクタ長が5のときには1.5ストップビット
		ビット1,0　キャラクタ長
		00 = 5ビット
		01 = 6ビット
		10 = 7ビット
		11 = 8ビット
03FCh	I/O	モデム制御レジスタ
		ビット7-5　つねに0
		ビット 4　8250B/16550を自己診断モードにする
		ビット 3　1 = 8250B/16550の割り込み要求信号を処理する
		ビット 2　1 = Heyesコマンド内蔵モデムの電源オンリセット
		ビット 1　1 = RS(RTS)信号をオンにする
		ビット 0　1 = ER(DTR)信号をオンにする
03FDh	IN	ラインステータスレジスタ

アドレス		機能
		ビット 7　1 = FIFOバッファ中のデータにPE,FE,OEがあった ビット 6　1 = 送信保持レジスタ・送信シフトレジスタが空 FIFOモード時は，送信FIFOレジスタ＋シフトレジスタが空 ビット 5　1 = 送信保持レジスタが空 FIFOモード時は，送信FIFOレジスタが空 ビット 4　1 = ブレーク信号を検知したとき ビット 3　1 = フレーミングエラー ビット 2　1 = パリティエラー ビット 1　1 = オーバーランエラー ビット 0　1 = 受信データレディ標識
03FEh	IN	モデムステータスレジスタ ビット 7　CD(DCD)のレベル ビット 6　CI(RI)のレベル ビット 5　DR(DSR)のレベル ビット 4　CS(CTS)のレベル ビット 3　CD(DCD)の変化 ビット 2　CI(RI)の変化 ビット 1　DR(DSR)の変化 ビット 0　CS(CTS)の変化
03FFh	I/O	スクラッチパッドレジスタ
1160h,1162h		フォントROMコントローラ（フォントROMをもっている機種のみ）

資料編4 割り込み一覧

DOS/V

割り込みタイプ		名前	
00h		0による割り算	
01h		シングルステップ割り込み デバッグ割り込み(80386+)	
02h		マスク不可割り込み(NMI)	
03h		ブレークポイント	
04h		オーバーフロー	
05h		画面印刷	
06h-07h		システム予約	
08h	8259 No.1 割り込み	タイマ割り込み (1秒間に18.2回)	IRQ0
09h		キーボード割り込み	IRQ1
0Ah		2番目の8259へ接続	IRQ2
0Bh		COM2/COM4割り込み	IRQ3
0Ch		COM1/COM3割り込み	IRQ4
0Dh		パラレルポート(AT:LPT2&LPT3,PS/2:LPT3)	IRQ5
0Eh		ディスケット(FDD)割り込み	IRQ6
0Fh		パラレルポート(AT:LPT1,PS/2:LPT1&LPT2)	IRQ7
10h	BIOS	ビデオBIOS	
11h		装置構成情報の取得	
12h		メモリサイズの取得	
13h		ディスケットディスクBIOS	
14h		シリアルポートBIOS	
15h		システムBIOS	
16h		キーボードBIOS	
17h		プリンタBIOS	
18h		予約済み(ROM BASICの起動)	
19h		システム再ブート	
1Ah		システムタイマBIOS	
1Bh	ユーザーフック用	Ctrl+Breakハンドラ	
1Ch		タイマ割り込みハンドラ	
1Dh		システムデータ　ビデオパラメータ	
1Eh		システムデータ　ディスケットパラメータ	
1Fh		システムデータ　8×8グラフィックスフォント	
20h		システム終了	
21h		MSDOS ファンクションコール	
22h		プログラム終了アドレス	
23h		Break出口アドレス	
24h		致命的エラー処理ルーチンベクタ	
25h		絶対ディスク読み取り	
26h		絶対ディスク書き込み	
27h		常駐して終了	
28h		DOSアイドル割り込み	
29h		DOS高速文字出力	
2Ah		ネットワーク関連で使用	
2Bh-2Dh		予約済み	
2Eh		コマンド処理プログラム(COMMAND.COM)	
2Fh		多重割り込み	
30h-3Fh		予約済み(31=DPMI関連) (33=MS MOUSE関連) (3F=Linker関連)	
40h-5Fh		予約済み (4A=アラーム割り込み)	
60h-66h		ユーザープログラムで使用可	
67h		EMS	

割り込みタイプ		名前	
68h-6Fh		予約済み	
70h		リアルタイムクロック割り込み	IRQ8
71h		IRQ9をIRQ2(INT 0Ah)にリダイレクト VGA(Video Graphics Array:N23 sxの場合)	IRQ9
72h	8259 No.2	H/W 割り込み	IRQ10
73h	割り込み	H/W 割り込み パワーマネージメント(N23sxの場合)	IRQ11
74h		マウス割り込み(PS/2)	IRQ12
75h		数値演算コプロセッサ割り込み	IRQ13
76h		ハードディスク割り込み	IRQ14
77h		ハードウェア割り込み	IRQ15
78h-FFh		予約済み	

ハードウェア割り込みは2個の8259を使用し，16レベルの割り込みがINT 08hとINT 70hの2か所に用意されています.

ハードウェア割り込みのIRQ9は，カスケードレベルIRQ2の割り込みレベルの置き換えとして定義されています.

添付ディスクについて

添付ディスクには，以下に示すサブディレクトリがあり，本文中で解説したプログラムのソースファイルおよび実行形式ファイルなどが含まれています．

本書のサブルーチン群を使用すれば，C言語から簡単にBIOS機能を使用したプログラムを作成することができます.また,ソースもすべて添付していますので，自由に改変することができます．

動作に関する保証・責任は一切行いません．また，著作権は私達が保持します．ただし，このプログラム自身を販売することを除き，商用使用および自社開発ソフトウェアへの組み込みなど一切自由とします．

ディレクトリFSWに含まれるフリーソフトウェアに関しては，著作権・再配布条件などは，中に含まれるドキュメントの指示に従ってください．

これらのプログラム，内容に関しては，NIFTY Serve FGALST内のDOS/Vプログラミング会議室でフォローしたいと思っています．

ルートディレクトリ

LLIB.LZH	ラージモデル用ライブラリ関数LHA
SLIB.LZH	スモールモデル用ライブラリ関数LHA
COMMON.H	C言語のタイプ定義など
RS232C.H	RS-232C用の定数定義
STD.INC	MASM用定数およびマクロ定義
LHA213.EXE	高圧縮書庫管理プログラム LHA
README.DOC	ドキュメント

PROGRAMディレクトリ　第1章関連プログラム

MAKEFILE	
STD.INC	
ISMODE.ASM	機種判定メインルーチン
ISMACHIN.ASM	機種判定ルーチン
ISMODE.OBJ	
ISMACHIN.OBJ	
ISMODE.COM	

VIDEOディレクトリ　第2章関連プログラム

MAKEFILE	
MAKEFILE.BCC	
TURBOC.CFG	
ROMBIOS.H	C言語ヘッダファイル
SDP.C	ファイルダンププログラム
SDP.OBJ	
SDP.EXE	

KEYBORDディレクトリ　第3章関連プログラム

MAKEFILE	
STD.INC	
SCANKEY.ASM	キー捜査コード表示プログラム
KEYTEST.ASM	Alt+1を内部処理するプログラム
FEPCTL.ASM	FEP制御ルーチン
FEPTEST.ASM	FEP制御テストプログラム
SCANKEY.OBJ	
KEYTEST.OBJ	
FEPTEST.OBJ	
SCANKEY.COM	
KEYTEST.COM	
FEPTEST.EXE	
FEPTEST.MAP	

RS232Cディレクトリ　第4章関連プログラム

MAKEFILE	
STD.INC	
RSDRV.INC	MASM用定数定義ファイル
IS16550.ASM	NS16550の判定ルーチン
RSDRV.ASM	割り込み処理ルーチン
SIGCTL.ASM	信号線制御関数群
BREAK.ASM	ブレーク信号送出ルーチン
COMMON.H	
RS232C.H	
RSMAIN.C	シリアルポートOPEN/CLOSEルーチン

RSDRV.OBJ
RSMAIN.OBJ
SIGCTL.OBJ
BREAK.OBJ
IS16550.OBJ
RSCOM.EXE

MOUSEディレクトリ	第5章関連プログラム
MAKEFILE	
STD.INC	
MOUSE.ASM	マウス制御関数群
MOUSE.H	
ROMBIOS.H	
MTEST.C	マウステストプログラム
MTEST.OBJ	
MOUSE.OBJ	
MTEST.EXE	

TIMERディレクトリ	第6章関連プログラム
MAKEFILE	
STD.INC	
COMMON.H	
TIMER.C	タイマ分周テストプログラム
REALTIME.ASM	タイマ分周処理ルーチン
BUZZ.C	ブザーテストプログラム
TIMER.EXE	
BUZZ.EXE	
BUZZ.OBJ	
REALTIME.OBJ	
TIMER.OBJ	

OTHERディレクトリ	第7章関連プログラム
MAKEFILE	
STD.INC	
DRIVE.ASM	ディスクアクセス関数
COMMON.H	
SINGLED.C	シングルドライブ対応ビューワー
SINGLED.EXE	
SINGLED.OBJ	
DRIVE.OBJ	

LLIB.LZH収録ファイル	ラージモデル用ライブラリ関数
MAKEFILE	TMRWAIT.ASM
ROMBIOS.L	ATTRIB.OBJ
STD.INC	BUZZER.OBJ
ATTRIB.ASM	CHGENV.OBJ
BREAKKEY.ASM	CURSOR.OBJ
BUZZER.ASM	FEPCTL.OBJ
CHGENV.ASM	GETSTR.OBJ
CURSOR.ASM	PUTSTR.OBJ
FEPCTL.ASM	GETVMOD2.OBJ
FONT.ASM	GETVMODE.OBJ
GETSTR.ASM	ISANSI.OBJ
GETVMOD2.ASM	ISMACHIN.OBJ
GETVMODE.ASM	REWRITE.OBJ
ISANSI.ASM	SCROLL.OBJ
ISMACHIN.ASM	PALETTE.OBJ
KEYIN.ASM	FONT.OBJ
KEYSFT.ASM	SETVSEG.OBJ
PALETTE.ASM	TEXTCTRL.OBJ
PUTKEY.ASM	BREAKKEY.OBJ
PUTSTR.ASM	KEYSFT.OBJ
REWRITE.ASM	KEYIN.OBJ
SCROLL.ASM	PUTKEY.OBJ
SETVSEG.ASM	TMRWAIT.OBJ
TEXTCTRL.ASM	ROMBIOS.LIB

SLIB.LZH収録ファイル	スモールモデル用ライブラリ関数
MAKEFILE	TMRWAIT.ASM
ROMBIOS.L	ATTRIB.OBJ
STD.INC	BUZZER.OBJ
ATTRIB.ASM	CHGENV.OBJ
BREAKKEY.ASM	CURSOR.OBJ
BUZZER.ASM	FEPCTL.OBJ
CHGENV.ASM	GETSTR.OBJ
CURSOR.ASM	PUTSTR.OBJ
FEPCTL.ASM	GETVMOD2.OBJ
FONT.ASM	GETVMODE.OBJ
GETSTR.ASM	ISANSI.OBJ
GETVMOD2.ASM	ISMACHIN.OBJ
GETVMODE.ASM	REWRITE.OBJ
ISANSI.ASM	SCROLL.OBJ
ISMACHIN.ASM	PALETTE.OBJ
KEYIN.ASM	FONT.OBJ
KEYSFT.ASM	SETVSEG.OBJ
PALETTE.ASM	TEXTCTRL.OBJ
PUTKEY.ASM	BREAKKEY.OBJ
PUTSTR.ASM	KEYSFT.OBJ
REWRITE.ASM	KEYIN.OBJ
SCROLL.ASM	PUTKEY.OBJ
SETVSEG.ASM	TMRWAIT.OBJ
TEXTCTRL.ASM	ROMBIOS.LIB

FSWディレクトリ	フリーソフトウェア
SETEV.LZH	SETEV
JBACK.LZH	JBACK
JB_16A.LZH	JBACK差分
$IASDMY.LZH	$IASDMY
WCDAT.LZH	WCDAT
CA565.LZH	CA(チャットアダプタ)
CASRC565.LZH	CAのソース

さくいん

[記号・数字・英字]

$DISP.SYS ……23
$DISPH.SYS ……25
$FONT.SYS ……23,171
$IAS.SYS ……90
55ミリ秒周期のタイマ割り込み ……158
8253 ……156
　～のカウントクロック ……157
8259A ……104
[Alt]+[SysRq]キー ……86
[Alt]キー ……87,90
ANSI.SYS ……12,34
ASYNC入出力 ……104
AT互換機のVRAM構成 ……22
[Break]キー ……85
CHEV ……17
CMOS-RAM ……163
CPUのフラグレジスタの方向フラグ ……106
[Ctrl]+[Alt]+[Delete]キー ……86
[Ctrl]+[Break]キー ……85
[Ctrl]キー ……88,90
DBCS ……18
　～コードジェネレータ ……81
　～ベクタテーブル ……18
[Delete]キー ……90
DISPS3.EXE ……25
DISPV.EXE ……25
DOS/V ……2
DOS/V BIOSモードの判定ロジック ……16
DOS/V Extension ……25,61
　DOS/V拡張モードテーブルの取得 ……63
　DOS/Vフォント品位の切り替え ……64
　DOS/Vテキスト密度の切り替え ……64
　DOS/Vモード設定の取得 ……64
　DOS/V Extensionの画面モード ……62
　高密度モードへの切り替え ……62
FIFO制御レジスタ ……110
FIFOモード ……113
FONTEX ……60
I/Oポートアドレス ……109
INS8250B ……107
[Insert]キー ……88
INT ……4
MOUSE.COM ……126
NS16550 ……107
[NumLock]キー ……89
[Pause]キー ……85,88
[PrintScreen]キー ……86
PS/2マウス対応インターフェース ……126
RS-232C ……5,115
　～の制御信号 ……116
RS/CSフロー制御 ……120
SBCS ……19
　～コードジェネレータ ……81
SETEV ……32
Super Drivers ……25,59
　新しくフォントを選択する ……61
　現在登録されているフォントの情報を得る ……60
　現在登録されているフォントの数を得る ……60
　現在選択されているフォントの名前を得る ……61
SVGA ……2
SVGA VESA BIOS ……73
SWITCH ……17
[SysRq]キー ……86
V-Text ……24
　～API ……32,62
　～インターフェース ……59
　非公開機能 ……65
VESA ……33
　～で規定しているビデオモード ……33
VGA ……2
　～グラフィック ……71
　～グラフィックコントローラ ……72
XGA ……2
XOFF ……119
XON ……119
XON/XOFFフロー制御 ……119

[あ]

アセンブラマクロ ……4
移動比率 ……129
色要素 ……51
英語モードと日本語モードの切り替え ……17
エスケープシーケンス ……34
エミュレートCGAテキストモード ……26
エミュレート拡張CGAテキストモード ……26

[か]

カーソル ……37
　～位置の設定 ……38
　～位置の文字と属性の読み取り ……39
　～形状・位置の取得 ……38
　～形状の設定 ……38
　～制御 ……37
　文字ブロックの読み取り ……40
各種機種・DOS判定ルーチン ……14
仮想VRAM ……44
可変高密度モード ……62
カラーパレット ……51

一括カラーレジスタ ……54
一括パレット ……52
オーバースキャンレジスタ ……53
個別カラーレジスタ ……53
個別パレット ……52
カラーレジスタ ……51
～の初期値 ……52
～の初期値（色系列別） ……53
漢字コード ……7
漢字シフト ……90
MS-KANJI ……92
かな漢字変換プログラム状況の設定 ……93
キーボード状況モードの設定 ……91
キーボード状況モードの読み取り ……90
現在のかな漢字変換プログラム状況の参照 ……92
～の制御 ……90
漢字の泣き分かれ ……45
画面印刷 ……86
画面制御 ……5,13
画面属性 ……27
画面表示 ……5
画面表示ドライバ ……24
キー入力バッファ ……82,99
～への書き込み ……99
キーボード ……5,78
～BIOS ……5
～インターセプトルーチン ……100
～からの入力 ……95
～ハードウェア割り込みハンドラ ……100
～のレイアウト ……79,80
～割り込み処理 ……81
キーリピート機能 ……97
キーボードの種類 ……78
グラフィックカーソル ……128
グラフィック画面 ……22
グラフィック処理 ……71
グラフィックモード ……26
罫線 ……9
コードページ ……19
高速スクロール ……49
高品位 ……25
～モード ……62
高密度 ……25
～モード ……32

[さ]

サウンド ……165
座標 ……127
システム時刻の設定 ……163
システムタイマ ……161
～の時刻カウントの読み取り ……161
システム制御ポートB ……167
システム日付の設定 ……162
シフトキー ……87
シフト状況 ……87
～の取得 ……87
シフト状態 ……87
～の制御 ……89
シリアルコントローラ ……107
シリアル通信 ……5,104,109,115
～I/Oポートアドレス ……108
～I/Oポートベースアドレス ……107
シリアルポート ……104
シリアルマウスインターフェース ……126
信号線の制御 ……115
受信FIFOバッファ ……119
受信データレジスタ ……119
受信データレディ割り込み ……111,113,119
スクロール ……46
スタックのオーバーフロー ……106
スピーカ ……165
セグメントレジスタの代入 ……4
走査コード ……96,100
～セット ……83
～の変換 ……100
送信FIFOバッファ ……120
送信保持レジスタ ……120
送信レジスタ空き割り込み ……111,113,120
送受信処理 ……121

[た]

タイパマティック ……97
～キー ……78
タイマ ……7
タイマチップ ……165
～へのカウンタ送出 ……166
タイマ割り込み ……158
～ハンドラ ……160
～ルーチンの解除 ……160
～ルーチンの設定 ……159
タイムアウト割り込み ……113
通信速度と分周値の関連 ……112
テキストカーソル ……129
テキスト画面 ……22
特殊キー ……85
特殊文字 ……9
トグルキー ……87
ドライブ制御 ……171

[な]

日本語表示のしくみ ……22

[は]

ハードウェアを操作 ……13

ハードウェアスクロール ……46
ハードウェアタイマ……7
　～割り込みハンドラ ……157
ハードウェア割り込み ……104
ハイテキスト ……24
半角フォント ……55
　英語モードの～ ……56
　日本語モードの～ ……56
パラレルポート ……170
　～I/Oレジスタ ……170
　～のI/Oポートベースアドレス ……170
パレットの初期値 ……51
標準モード ……62
非同期通信 ……104
ビデオBIOS ……5,12
　～呼び出しの機能番号 ……31
ビデオモード……22,26
　～番号 ……28
ファイルダンプ ……65
フォント ……55
　全角文字～パターンの設定 ……57
　半角文字～パターンの登録 ……58
　～ドライバ ……24
　～の高さ ……56
　～の登録 ……55
　～の幅 ……56
　～の読み取り ……55
　～の読み取り／書き込み ……60
　文字～パターンの取得 ……57
　文字ボックスと～ ……63
ブレーク信号 ……122
　～送出 ……122
プリンタポート ……170
プログラマブルインターバルタイマ ……156
プログラムの動作環境 ……14

[ま]

マウス……6,130
　カーソル位置とボタン状態の読み取り ……133
　カーソル位置のセット ……134
　カーソル移動範囲の設定ーX方向 ……137
　カーソル移動範囲の設定ーY方向 ……138
　カーソルの表示 ……133
　カーソルの非表示 ……133
　カーソル表示ページの取得 ……152
　カーソル表示ページの設定 ……151
　カーソル非表示域の設定 ……143
　グラフィックカーソルの形状設定 ……138
　代替ユーザー割り込み用トリガ ……147
　代替ユーザー割り込みルーチンのアドレスの取得 148
　代替ユーザー割り込みルーチンの設定 ……146
　代替割り込みルーチンに渡される要因と状態……148
　テキストカーソルの設定 ……139
　ドライバ状態の復元 ……146
　ドライバ状態の保管 ……146
　ドライバ状態保管用バッファサイズの取得……145
　ボタンを押した回数と最終位置の読み取り……135
　ボタンを離した回数と最終位置の読み取り……136
　～API ……130
　～BIOS ……6
　～BIOSの比較 ……6
　～インターフェース ……126
　～インターフェースの比較……6
　～カーソル ……128
　～感度の取得 ……151
　～感度の設定 ……151
　～機能の初期化 ……132
　～ドライバ機能の一覧 ……130
　～ドライバの使用禁止解除 ……153
　～ドライバの使用禁止設定 ……152
　～ドライバのソフトウェアリセット ……153
　～の移動距離 ……129
　～の移動距離の読み取り ……140
　～の移動比率の設定 ……143
　～のΔ ……129
　ユーザー割り込みの設定 ……140
　ユーザー割り込みルーチンの差し替え ……145
　ライトペンエミュレーション機能開始 ……142
　ライトペンエミュレーション機能終了 ……143
　割り込みルーチンに渡される要因と状態……141
マルチフォント環境 ……59
メークキー ……78
メークブレークキー ……78
文字列の書き込み ……42
文字コード ……96
文字出力 ……12
文字の書き込み ……41
　現在のカーソル位置へ文字を書き込む ……42
　現在のカーソル位置へ文字と属性を書き込む ……42
　テレタイプ式文字の書き込み ……41
モデムステータスレジスタ ……117
モデムステータス割り込み……111,113
モデム制御レジスタ ……116

[ら]

ラップラウンド処理 ……47
ラインステータスレジスタ ……119
ラインステータス割り込み……111,113
ライン制御レジスタ……111,112
リアルタイムクロック ……160
　～のアラーム解除 ……163
　～のアラーム設定 ……163
　～の時刻の読み取り ……162
　～の日付の読み取り ……162

リブート ……86
レジスタのポップ ……4

[わ]

割り込み許可レジスタ ……111
割り込み識別レジスタ ……119
割り込みの禁止 ……111
割り込みベクタ ……108
　～の書き換え ……112
　～の保存 ……112
割り込みマスクレジスタ ……114
割り込み呼び出し ……4

図一覧

図1.1　DOS/V BIOSモードの判定ロジック ……16
図1.2　日本語環境と英語環境の切り替え ……17

図2.1　AT互換機のVRAM構成 ……22
図2.2　DOS/V日本語表示処理のしくみ ……23
図2.3　画面属性 ……27
図2.4　カーソルの位置と形状 ……37
図2.5　ハードウェアスクロール ……47
図2.6　ラップラウンド処理 ……48
図2.7　日本語モード時の半角フォント ……56
図2.8　英語モード時の半角フォント ……56
図2.9　フォント読み取り／書き込み ……60
図2.10　DOS/V Extinsionの画面モードの関連 ……62
図2.11　文字ボックスとフォント ……63
図2.12　ファイルバッファの構造 ……66
図2.13　VGAグラフィックのしくみ ……71

図3.1　IBMでサポートするキーボードのレイアウト ……79
図3.2　OADGでサポートするキーボードのレイアウト ……80
図3.3　AXキーボード ……80
図3.4　J3100キーボード ……80
図3.5　キーボード割り込み処理 ……81
図3.6　キー入力バッファ構造 ……82

図4.1　シリアルポートコントローラブロック図 ……107

図5.1　マウスドライバ組み込みのメッセージ ……126
図5.2　画面と座標 ……127
図5.3　グラフィックカーソルのマスクパターン ……128
図5.4　テキストカーソルのマスクパターン ……129
図5.5　テキストカーソルのマスクビット構成 ……139
図5.6　ユーザー割り込み用トリガ ……141
図5.7　割り込みルーチンに渡される要因と状態 ……141
図5.8　代替ユーザー割り込み用トリガ ……147
図5.9　代替割り込みルーチンに渡される要因と状態 ……148

図6.1　タイマチップへのカウンタ送出方法 ……166
図6.2　システム制御ポートBのビット構成 ……167

図7.1　印刷データの流れ……171
図7.2　1ドライブ検査用BIOS作業エリア……172
図7.3　出力されるエラーメッセージ（A）……173
図7.4　出力されるエラーメッセージ（B）……173
図7.5　カレントドライブ管理エリア……174

表一覧

表0.1　マウスインターフェースの比較 ……6
表0.2　マウスBIOSの比較 ……6
表0.3　入れ替えられた22組の異体字 ……8
表0.4　追加された新字体漢字4文字／旧字は84区へ移動 ……8
表0.5　IBM選定文字に含まれていた2文字 ……8
表0.6　追加された特殊文字・罫線文字など69文字 ……8

表2.1　代表的な解像度 ……25
表2.2　ビデオモード番号 ……28
表2.3　VESAで規定しているビデオモード ……33
表2.4　DOS/Vエスケープシーケンスサポート表 ……35
表2.5　文字列ブロックの読み込み取り ……40
表2.6　文字列の書き込み ……43
表2.7　パレットの初期値 ……51
表2.8　カラーレジスタの初期値 ……52
表2.9　カラーレジスタの初期値（色系列別） ……53
表2.10 VGAグラフィック制御レジスタ ……71
表2.11 VESA機能一覧 ……72

表3.1　走査コードセット変換 ……83
表3.2　走査コードセット1 ……83
表3.3　BIOSワークエリアのキーボードシフト状況 ……88
表3.4　走査コード・文字コードの組み合せ ……96

表4.1　8259AのI/Oポートアドレス ……105
表4.2　シリアル通信I/Oポートベースアドレス ……107
表4.3　シリアル通信I/Oポートアドレス ……108
表4.4　FIFO制御レジスタ ……110
表4.5　割り込み許可レジスタ ……111
表4.6　ライン制御レジスタ ……112
表4.7　通信速度と分周値の関連 ……112
表4.8　RS-232Cの制御信号 ……116
表4.9　モデム制御レジスタ ……116
表4.10 モデムステータスレジスタ ……117
表4.11 割り込み識別レジスタ ……119
表4.12 ラインステータスレジスタ ……119

表5.1　マウスドライバがサポートする画面モード ……128

表5.2　マウスドライバ機能の一覧 …… 130
表5.3　初期化される内部変数 …… 132

表6.1　8253の技術資料 …… 156
表6.2　CMOS-RAMの構成 …… 164

表7.1　パラレルポートのI/Oポートベースアドレス …… 170
表7.2　パラレルポートのI/Oレジスタ …… 170

リスト一覧

リスト1.1 動作機種判定 ……15
リスト1.2 日本語環境と英語環境の切り替え関数 ……19

リスト2.1 ビデオモード取得・設定関数 ……31
リスト2.2 実際のビデオモード取得関数 ……32
リスト2.3 ANSI.SYS組み込み確認関数 ……34
リスト2.4 カーソル形状・位置の設定 ……38
リスト2.5 カーソル位置の文字と属性の読み取りを使った文字ブロックの読み取り ……40
リスト2.6 文字ブロックの読み取り機能を使用した文字ブロックの読み取り ……41
リスト2.7 テレタイプ式文字の書き込み ……42
リスト2.8 文字列の表示 ……43
リスト2.9 ビデオ初期化処理 ……44
リスト2.10 画面表示の更新 ……44
リスト2.11 画面の再表示 ……45
リスト2.12 文字列の表示2・文字の連続表示 ……46
リスト2.13 仮想VRAM内データの上方向スクロール ……49
リスト2.14 ビデオBIOSでのスクロール ……49
リスト2.15 パレットの保管・復元 ……54
リスト2.16 文字フォントパターンの取得と設定 ……58
リスト2.17 FONTEXのAPIアドレスの取得 ……60
リスト2.18 ファイルダンプの中心となるC部分 ……67

リスト3.1 ブレークキーの無効化 ……86
リスト3.2 BIOSワークエリアのキーボードシフト状況の取得 ……88
リスト3.3 キーボードシフト状況の取得 ……88
リスト3.4 キーボードシフト状態の変更 ……89
リスト3.5 FEPの制御関数 ……94
リスト3.6 キーボード入力処理 ……97
リスト3.7 キーコード指定によるキーボード入力処理 ……97
リスト3.8 キーボードタイプ速度の設定 ……98
リスト3.9 キーボードバッファの消去処理 ……98
リスト3.10 キー入力バッファへのデータの書き込み ……99
リスト3.11 キーボードインターセプトルーチンに渡される走査コード ……101
リスト3.12 キー走査コードを処理する ……101

リスト4.1 I/Oポートのベースアドレスと割り込み番号の決定 ……109
リスト4.2 NS16550の判定 ……110
リスト4.3 通信速度・キャラクタ長などの設定 ……112

リスト4.4　RS-232Cオープン処理 …… 114
リスト4.5　信号線関連の関数 …… 117
リスト4.6　送受信処理 …… 121
リスト4.7　ブレーク信号の送出 …… 123

リスト5.1　MOUSE.H …… 131
リスト5.2　マウス機能の初期化 …… 132
リスト5.3　カーソルの表示／非表示 …… 133
リスト5.4　カーソル位置とボタン状態の読み取り …… 134
リスト5.5　カーソル位置のセット …… 135
リスト5.6　ボタンを押した回数と最終位置の読み取り …… 136
リスト5.7　ボタンを離した回数と最終位置の読み取り …… 137
リスト5.8　ドラッグ処理 …… 138
リスト5.9　カーソル移動範囲設定 …… 138
リスト5.10 グラフィックカーソルの形状定義 …… 139
リスト5.11 テキストカーソルの形状定義 …… 140
リスト5.12 マウスの移動距離の読み取り …… 140
リスト5.13 ユーザー割り込み …… 142
リスト5.14 ライトペンエミュレーション制御 …… 143
リスト5.15 カーソル非表示域の設定 …… 144
リスト5.16 倍速境界値の設定 …… 144
リスト5.17 ユーザー割り込みルーチンの差し替え …… 145
リスト5.18 ドライバ状態の保管／復元 …… 146
リスト5.19 代替ユーザー割り込み …… 149
リスト5.20 マウス感度の設定 …… 151
リスト5.21 マウス感度の取得 …… 151
リスト5.22 カーソル表示ページの設定 …… 152
リスト5.23 カーソル表示ページの取得 …… 152
リスト5.24 マウスドライバの使用禁止設定 …… 153
リスト5.25 マウスドライバの使用禁止解除 …… 153
リスト5.26 マウスドライバのソフトウェアリセット …… 153

リスト6.1　タイマ割り込みの使用 …… 157
リスト6.2　55ミリ秒より短い間隔の割り込み …… 159
リスト6.3　タイマウェイト関数 …… 161
リスト6.4　スピーカのオン／オフの方法 …… 167
リスト6.5　ブザーを鳴らす …… 167

リスト7.1　1ドライブ検出方法 …… 172
リスト7.2　SetDiskette()関数 …… 174

●著者略歴

杉浦 明美（スギウラ アケミ）
DOS/V関連を中心とした，プログラム開発および月刊『ドクター・ドブズ・ジャーナル日本版』などに記事を執筆．
現在，㈲ゼータ・システムズを経営．
主人が，NIFTY Serve FGALSTのSYSOPをしており，DOS/V関連のフリーソフトウェアをいろいろと作成しているので，技術的監修はすべて主人に依頼している．

柴﨑 忠生（シバザキ タダオ）
AT互換機，FM-R，PC98など数多くのパソコンを仕事で使う．
現在，㈲マイクロダイナミクスを経営．
現在は次世代OSとしてOS/2に興味がある．

Programmer's Page
DOS/Vプログラミング技法

1993年6月1日　初版第1刷発行
1994年1月15日　初版第2刷発行

著　者　杉浦明美，柴﨑忠生
発行者　長廻健太郎
発行所　株式会社翔泳社
〒107　東京都港区北青山3-10-18
北青山本田ビル
出版事業部編集部　03-5467-3777
出版事業部販売部　03-5467-0361
装　丁　STARKA
組　版　株式会社アビック
印　刷　株式会社サンニチ印刷

ISBN4-88135-034-X

株式会社 翔泳社 定価2800円（本体2718円、税82円）
ISBN4-88135-034-X C3055 P2800E

programmer's page

DOS/Vプログラミング技法

杉浦明美・柴﨑忠生 共著

3.5"
2DD FD付

SE
SHOEISHA

本書の構成

第０章・はじめに
第１章・DOS/Vでのプログラミングとは
第２章・画面制御編
第３章・キーボード編
第４章・RS-232C編
第５章・マウス編
第６章・タイマ編
第７章・その他の周辺機器編
資料編・BIOS割り込み一覧
BIOSワークエリア
I/Oポート
割り込み一覧
添付ディスクについて